Une approche qualitative pour la prise de décision sous contraintes non-fonctionnelles dans le cadre d'une composition agile de services

Pierre Châtel

Université Pierre et Marie Curie – Paris 6

LIP6 – UMR 7606
Département RSR – Équipe MoVe

Thales Communications France
Service SC2

Une approche qualitative pour la prise de décision sous contraintes non-fonctionnelles dans le cadre d'une composition agile de services

THÈSE

présentée et soutenue publiquement le 5 mai 2010

pour l'obtention du

Doctorat de l'université Pierre et Marie Curie – Paris 6

(spécialité informatique)

par

Pierre Châtel

Composition du jury

Rapporteurs :	Luis Martínez López	Professeur, Université de Jaén, Espagne
	Laurence Duchien	Professeur, Université Lille 1
Directeurs :	Jacques Malenfant	Professeur, Université Pierre et Marie Curie – Paris 6
	Isis Truck	Maître de Conférences, Université Vincennes - Saint-Denis – Paris 8
Examinateurs :	Florence Sèdes	Professeur, Université Paul Sabatier – Toulouse 3
	Emmanuel Chailloux	Professeur, Université Pierre et Marie Curie – Paris 6
Encadrant industriel :	Hugues Vincent	Responsable de Laboratoire R&D, Thales Communications France

École Doctorale d'Informatique, Télécommunication et Électronique de Paris

Remerciements

Ces remerciements sont adressés à toutes les personnes m'ayant assisté dans mon travail et soutenu au quotidien, par leurs actions ou leur simple présence, au cours de ces années de thèse. J'espère avoir l'opportunité de tous les remercier chaleureusement en personne, au delà du cadre formel de ce chapitre. Cependant, *"contrairement à la parole, l'écrit ne s'évanouit pas dans l'air"*, et c'est dans cet esprit que je tiens à remercier tout particulièrement les personnes suivantes.

En tout premier lieu, ma gratitude est tournée vers mes deux directeurs de thèse, Jacques Malenfant et Isis Truck, dont le professionnalisme et la pugnacité sans faille ont permis de dresser le chemin suivi par cette thèse et de m'y accompagner. J'ai pu, avec eux, partager de très enrichissantes discussions, souvent animées, qui ont ouvert de nouveaux horizons à ma curiosité scientifique. Ils sont devenus pour moi, à l'issue de ces trois années, autant des maîtres à penser, que des amis sur lesquels je sais pouvoir compter.

Par ailleurs, je suis honoré par la participation de Laurence Duchien et Luis Martínez López au jury de cette thèse, en tant que rapporteurs ; ainsi que celle de Florence Sèdes et Emmanuel Chailloux en leur qualité d'examinateurs.

Je souhaite aussi exprimer ma plus vive reconnaissance à Pascal Llorens, pour m'avoir accueilli au sein de son service de recherche et développement à Thales Communication, ainsi qu'à mon encadrant industriel Hugues Vincent qui a su me soutenir et m'a permis d'acquérir, par sa rigueur naturelle, une double culture : scientifique et industrielle.

Je tiens aussi à remercier mes collègues du service SC2 pour leur support moral et intellectuel, en particulier ceux avec lesquels j'ai pu construire de véritables liens d'amitié pendant toutes ces années : Antoine Leger et Tran Huynh, pour ne citer qu'eux. Une thèse étant avant tout une aventure humaine, ce manuscrit est aussi un peu le leur.

Finalement, mes sentiments les plus chers vont à mon compagnon, ma famille et mes amis : Gilles, par ta présence au quotidien et tes encouragements, c'est à toi que je dois d'avoir réussi à passer sans encombre l'ultime ligne droite que représente ma dernière année de thèse. J'éprouve aussi le plus grand respect envers mes parents, qui ont toujours soutenu mes choix sur le long terme, et m'ont aidé à les atteindre : leur support a été essentiel. Pour finir, une pensée toute particulière pour mes amis. Merci à vous tous : Claire, Aurélien, Anneli, Pierre-Arnaud, Karine, Gabriel, Laure, Fabien, Franco, Olga, et les autres.

"Broadly speaking, the history of software development is the history of ever later binding times..."
- M. Halpern, Encyclopedia of Computer Science [Halpern, 1993]

Table des matières

Table des figures

Extraits de programmes

Chapitre 1

Introduction

Les systèmes informatiques, en particulier répartis, sont en constante évolution. Or, leur complexification va souvent de pair avec cette évolution. De ce fait, la pérennité d'une infrastructure logicielle, en entreprise comme sur le Web, est directement liée à sa capacité à intégrer de multiples altérations de contexte afin d'accompagner ces mutations.

Les architectures déployées doivent donc faire preuve de flexibilité, et ce dans un contexte de rationalisation des développements. Cette thèse a ainsi pour vocation de lever et résoudre certains verrous scientifiques et techniques à la mise en œuvre d'une réelle agilité des systèmes répartis dits complexes, condition *sine qua non* de leur pérennité, tout en s'attachant à préserver, ou même améliorer, leurs performances.

Par ailleurs, tel que nous le verrons dans la section suivante, un certain nombre de paradigmes de programmation ont successivement posé le terreau conceptuel et technologique permettant à nos travaux de germer. Travaux qui, tout en s'inscrivant dans l'héritage de ces paradigmes, ont alors pour objet d'apporter une réponse inédite aux nouveaux défis posés par ces architectures réparties complexes, et plus particulièrement à la composition dynamique de services sous contraintes non-fonctionnelles.

1.1 De l'approche à objets au paradigme de service

Un des premiers paradigmes à s'être révélé pertinent pour répondre à ces nouveaux défis est la Programmation Par Objet ("Object Oriented Programming") [Jacobson, 1991], dont on peut retracer les origines jusqu'au début des années 1960. Ce paradigme consiste en la définition et l'assemblage de briques logicielles appelées objets ; un objet représentant le plus souvent un concept, une idée ou une entité physique. La force de l'approche à objets réside donc dans sa capacité à réconcilier les plans logiciel et utilisateur par la modélisation objet, et par là même de simplifier la mise au point de systèmes informatiques complexes. Son adoption massive n'a cependant commencé qu'au début des années 1990, poussée ensuite par la visibilité accrue des Processus Unifiés [Jacobson et al., 1999], ces méthodologies de développement logiciel dont le "Rational Unified Process" (RUP) d'IBM, est l'une des plus célèbres incarnations.

Par la suite, le paradigme de composant [Szyperski et al., 1999], implanté notamment par les technologies CORBA ou JEE, s'est construit autour des principaux concepts objets, tels l'encapsulation des fonctionnalités et des données, auxquels s'ajoute notamment la contractualisation des interactions, la composition par des tiers et la répartition des composants. Un composant possède ainsi des interfaces bien spécifiées et peut être déployé indépendamment du reste de l'application. La Programmation Orientée Composant (POC) n'est d'ailleurs pas sans similitudes avec l'approche à objets, puisqu'elle revient à utiliser une approche comparable, non pas directement au sein du code, mais au niveau de l'architecture générale du logiciel : elle a ainsi permis le regroupement cohérent et réutilisable des objets.

Favorisé notamment par l'irrésistible montée en puissance d'Internet, le monde de la programmation répartie est, aujourd'hui encore, en cours de mutation avec l'adoption progressive du paradigme de service et la mise en œuvre des architectures fondées sur ces services ("Service Oriented Architectures", ou SOA)) [Papazoglou et Georgakopoulos, 2003].

Les SOA mettent en avant un couplage théoriquement lâche entre client et fournisseur de service. Elles découplent ainsi une application répartie en deux couches. Une couche d'offre de services spécifiés par leurs interfaces et éventuellement des informations de nature sémantique dans un annuaire de services. La seconde couche est la réalisation physique des services sur des machines connectées à Internet et accessibles *via* des protocoles standards.

La programmation d'une application est alors vue comme la description de la combinaison d'un ensemble de services, d'une granularité équivalente à celle des composants, dont la composition lors de l'exécution permet de réaliser le calcul voulu par l'application. Cette composition, le plus souvent appelée "orchestration", va alors s'effectuer sur la base d'informations portées par un processus métier dont l'exécution va d'abord demander de lier les appels abstraits de service à des réalisations de services compatibles, fournies par un annuaire. Il s'agit donc de faire le lien entre une spécification abstraite des fonctionnalités requises dans le processus afin de constituer l'application, et les fonctionnalités disponibles sur le réseau *via* leur implantation concrète sous forme de services.

D'un point de vue technologique, une autre caractéristique importante des SOA tient dans le fait qu'elles reposent sur des protocoles et des représentations de données standards, le plus souvent utilisant des langages fondés sur XML, facilitant l'interopérabilité. C'est dans ce cadre qu'est concrétisé le concept de service Web ("Web service"), tel que défini par l'organisme de standardisation W3C [1], et la cohorte de technologies et langages associés, dont SOAP et WSDL sont les plus célèbres représentants. L'un constituant un protocole d'échange de messages vers les services Web, l'autre un langage de description d'interfaces ou offres de services.

Un système informatique fondé sur une architecture SOA est ainsi disponible sous la forme de services réutilisables qu'il est possible de découvrir et composer dynamiquement avec un couplage lâche. Il se distingue dès lors des solutions totalement intégrées plus traditionnelles, de type boîte noire, que sont ERP ou autres progiciels. Cette architecture veut ainsi répondre aux besoins de flexibilité, réutilisabilité, et d'adaptabilité rapide mis en avant dans les grands systèmes logiciels actuels et souhaités par les entreprises. C'est pour ces raisons et l'adoption de plus en plus importante de ces technologies dans les entreprises, que nous fonderons notre approche sur le paradigme de service.

1. *World Wide Web Consortium*

1.2 Genèse d'une problématique

Plongé dans un contexte industriel, ce travail de thèse trouve écho dans un vaste cadre d'applications s'étendant des Systèmes de Systèmes ("Systems of Systems", SoS) classiques, aux applications de type C2 ("Command and Control", Commande et Contrôle) déployées dans le cadre d'opérations tactiques militaires ou de gestion de crise, ainsi qu'à l'informatique omniprésente ("Pervasive Computing").

Dans tous ces domaines, on est amené à gérer la composition dynamique de services avec une forte contrainte de qualité et un minimum de garanties. Les aspects non-fonctionnels qui influent sur le niveau de service rendu ou perçu, tels les délais de réaction des applications, ou plus généralement la Qualité de Service ("Quality of Service", QoS) technique ou métier offerte par ces applications, prennent une alors importance cruciale. Les architectures fondées sur les services doivent donc être appareillées pour répondre à ces différentes exigences. D'autres caractéristiques sont aussi essentielles dans ces applications :

- La mobilité, qui rend l'apparition et la disparition de services très fréquente ;
- La redondance, qui fait en sorte qu'un même service peut à tout instant être rendu par de nombreuses réalisations physiques mais qui ont toutes une certaine probabilité d'être détruites (cette probabilité varie également fortement selon le type et la phase opérationnelle dans laquelle se retrouve le dispositif, engagé ou non) ;
- L'hétérogénéité, un même service peut être rendu par de nombreux dispositifs très différents. Par exemple, une imagerie numérique peut être rendue par la caméra à basse résolution intégrée dans le casque d'un fantassin ou encore la caméra à haute résolution d'un char de reconnaissance.

De ces diverses caractéristiques émerge la problématique générale de la mise en œuvre d'une composition de servicesComposition de services agile, qui, pour être utile, doit prendre en compte l'hétérogénéité du contexte ainsi que les multiples contraintes non-fonctionnelles qui s'y rapportent.

1.3 Objectifs de recherche et approche poursuivie

Plusieurs obstacles viennent jalonner le cheminement vers une composition agile de service. Ainsi, les nombreuses propriétés non-fonctionnelles à traiter peuvent s'étendre des caractéristiques techniques d'un service (comme sa disponibilité ou bande passante offerte), à des propriétés non-fonctionnelles de plus haut niveau qui ne sont pas définies sur un ensemble fini, connu à l'avance, de dimensions de QoS. Ces dernières sont intimement liées au domaine de compétence (ou domaine"métier") du service. De surcroît, dans la vaste majorité des systèmes répartis à base de processus métiers, le lien avec les producteurs de services nécessaires est fixé, pour des raisons techniques ou humaines, de manière statique, *via* l'indication de l'emplacement physique des services sur le réseau, ou rigide, *via* la syntaxe. Dans les deux cas de figure, ceci nécessite la connaissance préalable des services disponibles au moment de l'écriture du processus.

C'est pourquoi, afin notamment de palier à ce manque de souplesse et s'aligner sur les besoins réels des systèmes répartis modernes en termes d'agilité, tels que mis en avant par la problématique de ce manuscrit, nous poursuivons les objectifs suivants dans le cadre de cette thèse :

Premier objectif : favoriser la réactivité des systèmes répartis complexes

Les SOA Sémantiques ("Semantic SOA", ou SSOA) [Vitvar et al., 2007], évolution récente des architectures SOA se situant à la croisée du Web Sémantique [Berners-Lee et al., 2001] et de la technologie des services Web [Papazoglou, 2008], proposent des outils à même de faciliter la description de propriétés métiers, fonctionnelles ou non, des services. Le premier d'entre eux consiste à permettre la définition et la réutilisation de concepts de haut-niveau au sein d'ontologies métiers, établies spécifiquement pour chaque domaine considéré. A partir de ces concepts, il est possible d'annoter sémantiquement les offres et requêtes de services, et d'en relever ainsi le niveau d'abstraction [Peer, 2002, Eberhart, 2004, Sivashanmugam et al., 2003, Patil et al., 2003].

On propose alors ici l'utilisation de ces nouvelles architectures SSOA car elles mettent en avant, par le détachement du niveau syntaxique et technique des services qu'elles proposent, un niveau élevé de découplage entre clients et fournisseurs de service. Par voie de conséquence, elles vont permettre la publication et la cohabitation d'une grande quantité de services répandant, pour certains d'entre eux, à un même besoin fonctionnel, mais exhibant le plus certainement des caractéristiques non-fonctionnelles (de Qualité de Service) bien distinctes.

Afin de tirer au mieux parti de la propriété de couplage faible offerte par les SSOA, et pour atteindre ce premier objectif en termes de réactivité des systèmes répartis complexes, nous proposons la mise en place d'un mécanisme de *liaison tardive de services* lors de l'exécution d'un processus métier, sur la base des valeurs courantes de QoS des services.

D'un point de vue technique, on se fonde alors sur une architecture logicielle à deux niveaux abstraits :

- Un niveau de filtrage des services puis d'exécution du processus métier. Le filtrage étant l'étape statique par laquelle des services disponibles sont associés, ou non, au processus à exécuter en fonction de leur pertinence par rapport aux stricts besoins fonctionnels et non-fonctionnels qui y sont exprimés.

- Et un niveau de contrôle du niveau d'exécution et de prise de décision de liaison en fonction de la disponibilité effective des services précédemment filtrés au moment de l'appel ainsi que de leurs *valeurs courantes de QoS* obtenues par un canevas externe de supervision. C'est ce niveau qui aura la charge de prendre les meilleures décisions de liaison entre les services disponibles et le processus en cours d'exécution, en fonction de critères définis par l'utilisateur.

Nous suivons en cela l'approche des langages réflexifs, qui découpent une telle architecture en un niveau de base (exécution) et un niveau *méta* de contrôle du niveau de base. Dans le cadre de cette thèse, on s'attachera ainsi tout particulièrement à la gestion des *moments tardifs de liaison* entre processus et services lors de leur composition, de manière à pouvoir notamment obtenir les valeurs effectives de QoS des services les plus "fraîches".

Second objectif : améliorer la performance des systèmes répartis complexes

Si le mécanisme de liaison tardive de services mise en œuvre par le premier volet de notre approche permet effectivement d'effectuer une décision éclairée lors du choix d'un service pour le lier à un processus, il reste encore à définir les critères qui là guideront, sur la base des valeurs courantes de QoS, cette décision vers une d'optimisation locale de l'exécution de ce processus.

Cependant, et à plus forte raison encore dans les SSOA, la notion de *performance* possède par nature de multiples facettes et reste dépendante du domaine métier de l'application : elle ne peut donc uniquement reposer sur un ensemble fixe et pré-établi des règles absolues. Par conséquent il sera nécessaire d'offrir à l'utilisateur les moyens de caractériser indirectement, pour son métier, cette notion de performance de manière à pouvoir ensuite l'intégrer au cœur du processus décisionnel de liaison.

Le deuxième volet de notre approche consiste alors à mettre en place un modèle qualitatif de préférences utilisateur employé pour maximiser l'utilité des liaisons entre producteurs et consommateurs de services.

On propose ainsi un formalisme qui va permettre au programmeur d'un processus métier d'exprimer ses "politiques de décision" dans la liaison et l'utilisation des services physiques en fonction des valeurs courantes de Qualité de Service. Ces *préférences utilisateur*, vont ainsi permettre de qualifier une relation d'implication entre les valeurs effectives des propriétés non-fonctionnelles d'un service et son degré transitoire d'utilité, par rapport à ses pairs, dans la liaison au processus métier.

La notion de performance, en tant qu'objectif à atteindre lors de la composition de services, s'exprime alors au travers de celle, moins subjective et plus facilement manipulable, d'*utilité des services* dans la composition ; et par extension de l'utilité d'un processus métier dans son ensemble, comme somme de toutes ses utilités locales. La bonne marche d'un système réparti, telle que souhaitée par l'utilisateur lors de la définition de ses préférences sur les services, se retrouve ainsi conditionnée à la capacité du processus d'orchestration de diriger ses choix de liaison en fonction des subtilités propres à son domaine d'exécution.

Troisième objectif : s'assurer de la simplicité et généricité de l'approche proposée

Le dernier défi de cette thèse, et non des moindres, sera de faire en sorte que les formalismes et réalisations techniques issues de la mise en œuvre de notre approche, restent simples d'utilisation et suffisamment génériques, afin d'être aisément adaptables à chaque domaine d'application métier et d'être en mesure d'accompagner leurs évolutions.

De fait, en l'absence dans notre contexte d'experts de l'élicitation de préférences, et dans l'optique de répondre à cet objectif de simplicité, ces dernières présenteront la double caractéristique d'être d'une part *qualitatives*, afin de gérer au mieux les imprécisions et incertitudes inhérentes à la modélisation de ce type de préférences, et d'autre part de permettre l'expression d'éventuelles mais potentielles contradictions entre plusieurs propriétés non-fonctionnelles offertes par un même service (par exemple, et très schématiquement, le *prix* vs. la *qualité*).

1.4 Contributions scientifiques

Les contributions scientifiques de cette thèse s'organisent tout naturellement autour des objectifs de recherche précédemment définis et vont ainsi venir se cristalliser sous trois dénominations distinctes : on va alors parler de composition *active*, *utile* et finalement *agile* des services dans le contexte SSOA qui est le notre.

En effet, la pleine réalisation du premier objectif en termes de réactivité des systèmes répartis complexes demande d'adapter l'approche proposée au cas singulier des SSOA et d'y intégrer des notions de décision qui ont peu été étudiées à ce jour. Par conséquent, **une première contribution apportée au domaine de la composition de services consistera à mettre en place une approche *active***, en plusieurs étapes, capable à différents moments (statiques et dynamiques) de filtrer puis sélectionner des services en se fondant sur leurs caractéristiques non-fonctionnelles. Cette contribution sera détaillée dans le chapitre 5 du manuscrit.

En réponse aux problématiques de performance précédemment évoquées, **une seconde contribution scientifique résidera dans la mise en œuvre d'une composition *utile* de services**, dans la mesure où elle cherche à maximiser l'utilité des liaisons processus/service en exploitant les particularités de ce nouveau formalisme de préférences utilisateur. Cette contribution sera détaillée, quant à elle, dans le chapitre 6 du manuscrit.

Finalement, **ces deux approches actives et utiles vont venir se combiner au sein d'une seule et même contribution en termes de composition *agile* de services**. Présentée dans le chapitre 7, elle consiste notamment à exploiter conjointement, au sein d'une orchestration de services fondée sur le langage BPEL, la liaison tardive des services et notre nouveau modèle qualitatif de préférences utilisateur ; ainsi que d'introduire des notions plus avancées telle la gestion de la Qualité de Service globale des processus métier lors des décisions tardives de liaison.

1.5 Organisation du document

Ce manuscrit s'articule autour de deux parties : dans un premier temps (cf. partie I) on effectue un *contexte et état de l'art* de manière à pouvoir positionner nos travaux par rapport à l'existant et identifier les éventuelles lacunes technologiques ou conceptuelles nécessitant d'être comblées. Cette première partie est accompagnée de la présentation du *cas d'utilisation* (cf. chapitre 4) dans lequel nos travaux vont être illustrés. Puis, en partie II, on aborde successivement *nos contributions* en terme de composition active, utile et agile de services (cf. chapitre 5, 6 et 7), tout en indiquant les correspondances possibles avec le cas d'utilisation, mais sans s'y limiter . Ces première contributions sont suivies d'une *formalisation* du modèle de préférences LCP-net (cf. chapitre 8) que nous avons introduit pour la composition utile, ainsi que de leur *mise en œuvre* technique (cf. chapitre 9). La présentation de notre *conclusion ainsi que des perspectives de nos travaux* closent finalement ce manuscrit.

Première partie

Contexte et Etat de l'Art

Chapitre 2

Contexte technologique

Sommaire

DANS CE CHAPITRE, on pose la base conceptuelle et terminologique nécessaire à la bonne compréhension de nos travaux. Ces derniers font en effet appel à de nombreuses notions issues de domaines en constante évolution. Il est ainsi important de garder à l'esprit le contexte industriel de notre recherche et le cadre technologique précis qui en découle. Il ne s'agit donc pas ici d'effectuer une description exhaustive de ces différents domaines, qu'il s'agisse de nouvelles approches, techniques, ou autres paradigmes de programmation ; mais bien d'en effectuer une prise de vue à un instant précis, celui de la sphère technique dans laquelle nos travaux évoluent.

Ce travail de thèse s'inscrit dans la lignée des travaux menés conjointement par Thales, Alcatel CIT Research and Innovation, Bull, Nokia, Schneider Electric et Vodafone dans le cadre du projet ITEA S4ALL [2] ("Services for All") dont le but est d'explorer les solutions techniques et consolider celles préexistantes permettant la mise en place d'un monde de services orientés utilisateurs simples à créer, partager et utiliser.

Mais il est aussi lié à certaines des problématiques rencontrées au cours du projet CARROLL [3]DYONISOS (INRIA Arles, CEA, Thales) dans le domaine des architectures fondées sur les

2. http ://www.itea-office.org/
3. http ://www.carroll-research.org/fr

services et des "systèmes de systèmes" dont le but est de permettre la composition dynamique de services et l'exécution de *"workflows"*.

Pour finir, le projet SemEUsE [4] a servi de substrat principal pour l'implantation des contributions scientifiques avancées dans ce manuscrit. Ce projet s'attache à la mise au point d'un "bus de service sémantique". Les objectifs suivants ont alors été définis : l'implantation d'applications omniprésentes, flexibles et fiables ; l'utilisation de services Web sémantiques (cf. 2.1.3) disponibles à l'exécution ; et l'exploitation d'offres de Qualité de Service métier de haut niveau (cf. 2.3) pour l'implantation des applications. Projet de trois ans, financé par l'Agence Nationale de la Recherche (ANR) et commencé en février 2008, il n'est donc pas encore clos au moment du dépôt de ce manuscrit. SemEUsE intègre les contributions de quatres partenaires académique (LIP6, INSA, INRIA, INT) et de trois partenaires industriels (Thales, Orange Labs, EBM Websourcing) au sein d'un consortium unique.

Afin d'accompagner le lecteur au travers de ces nombreux prérequis, nous nous attacherons dans un premier temps à détailler les concepts liés à l'Approche Orientée Services dans la section 2.1, ainsi nous nous pencherons tout particulièrement sur les technologies de services Web. Dans un second temps, en 2.2, nous présenterons les Processus Métier et la technologie BPEL qui sont à la base notre implantation. Pour finir, nous nous attarderons dans la section 2.3 sur les notions de propriétés et contraintes non-fonctionnelles de services ainsi que sur les travaux les plus connus les mettant en œuvre.

2.1 Architectures Orientées Services

On regroupe sous la dénomination d'Architecture Orientée Services ou *"Service Oriented Architectures"* (SOAs), l'ensemble des architectures mettant en avant la notion abstraite de *service* pour la mise en œuvre d'une réalisation logicielle, répartie ou non.

De fait, la propriété de *couplage lâche* entre les principales entités de l'architecture est particulièrement recherchée dans les SOAs. Elle favorise des systèmes s'appuyant sur des couches de médiation, imposées par des standards, capables d'exhiber un certain niveau d'interopérabilité au niveau de leurs communications. D'un point de vue technique, ce couplage lâche induit alors d'effectuer le moins d'hypothèses possibles lors de la description des différentes entités en interaction. Dans le cas des SOAs, cette approche est rendue possible par la cohérence des normes communément utilisées, avec notamment l'utilisation de modèles d'échanges pivots des données.

Les architectures orientées services sont ainsi le fruit d'une lente maturation technologique s'appuyant sur les travaux précurseurs à la base des notions d'objets et de composants. Une compréhension préalable de ces concepts est nécessaire à l'obtention d'une plus fine appréhension des SOAs.

2.1.1 Concepts de base

Objet

La complexification des systèmes logiciels a entraîné dans son sillage l'ingénierie logicielle vers un objectif de rationalisation de la production. Ainsi, de par leur évolution constante et l'augmentation

4. "Sémantique pour bus de services" : http ://www.semeuse.org/

de leur distribution sur le réseau, ces systèmes nécessitent la mise en place de paradigmes permettant d'assurer un certain niveau de qualité et fiabilité, tout en offrant une plus grande flexibilité et réutilisabilité.

L'approche à objets [Jacobson, 1991] s'est indubitablement révélée comme un standard *de facto* parmi les différents paradigmes de développement logiciel. Cette approche a été rendue possible par l'adoption intrinsèque du paradigme objet au cœur de langages tels Smalltalk, Java ou, dans une moindre mesure, C++. L'innovation portée par les objets a consisté à regrouper sous un même concept un ensemble de principes préexistants tels que :

- *l'encapsulation*, l'interface d'un objet est bien distincte de son implantation, et son comportement exprimée via les méthodes de cette interface ;
- *l'héritage*, c'est-à-dire de la capacité à définir les caractéristiques d'un objet par extension ;
- *le polymorphisme*, il est possible d'interagir de la même façon avec des objet de formes différentes ;

Composant

Par rapport aux objets, le *paradigme de composant* représente une augmentation du niveau d'abstraction lors de la conception logicielle. En ce sens, le composant ne remplace pas l'objet, il se construit même bien souvent autour, dans les langages de programmation courants (par exemple Java). Il répond d'un besoin concret car, si l'utilisation exclusive d'objets lors de la programmation de systèmes simples semble raisonnable, la programmation d'applications réparties de plus grande ampleur nécessite une augmentation du niveau d'abstraction de leurs briques de base, afin de maintenir leur complexité relativement appréhendable.

Un composant dispose d'interfaces spécifiées à l'aide de contrats, dans l'optique d'une composition ultérieure à sa conception, par des entités tierces. L'implantation de ce paradigme par de nombreuses technologies telles que les Enterprise Java Beans (EJB) de Sun, COM de Microsoft, ou FRACTAL du consortium OW2, a permis le leur large diffusion. Ces technologies déchargent le plus souvent le développeur de la gestion de certaines problématiques de mise en œuvre telles que le support de la sécurité, de la persistance et des transactions, pour ne citer qu'elles.

Ces mécanismes de support à l'exécution des composants seraient difficiles à mettre en place dans des environnements plus faiblement couplés (tels les services, détaillés à la sous-section suivante). La contre-partie est que les composants sont souvent bien trop liés à leurs plate-formes de support respectives pour être efficacement interopérables. Il s'agit pourtant là d'une de leurs caractéristiques les plus fondamentales. Se pose alors dans ce contexte la question de la méthodologie à utiliser pour assembler les composants disponibles ; question à laquelle la notion d'architecture, telle que définie par ces différentes technologies, tente de répondre.

Service

Dans la lignée de l'approche à objets ou à composants, le paradigme de service tend à fournir un niveau d'abstraction encore plus élevé lors du développement de systèmes informatiques en général, et des applications réparties en particulier. Cette augmentation du niveau d'abstraction passe par

une adoption poussée du concept d'encapsulation des fonctionnalités et de celui de la réutilisabilité des services existants. Les services répondent ainsi directement aux problématiques d'intégration logicielle dans les applications réparties modernes.

Tout comme un composant, un *service* est défini comme étant une unité logicielle autonome. Ce qui est d'autant plus vrai pour les services Web qui sont définis comme étant *sans état* (*"stateless"*). Les services se distinguent cependant des composants dans la mesure où leur définition met en avant une *indépendance poussée* par rapport aux plate-formes, un *couplage faible* (ou "lâche") et une *interopérabilité élevée*. Cette *autonomie* relègue les considérations purement technologiques au second plan, les services étant indépendants du contexte de leur utilisation ainsi que de l'état courant des autres services. Par ailleurs, leur propriété de faible couplage permet de mettre en œuvre une orchestration des services : cette approche introduit une plus grande flexibilité dans le choix des services à composer ainsi que dans le moment effectif de leur liaison. C'est ainsi que, dans nos travaux, nous cherchons à promouvoir une approche de *liaison tardive* des services lors de leur orchestration, de manière à exploiter plus avant cette flexibilité dans certains contextes particulièrement dynamiques.

De surcroît, les opérations courantes sur un service (comme sa mise au point, sa publication, ses interactions) s'effectuent toutes via des standards technologiques afin de faciliter leur collaboration à large échelle ; le cas d'application le plus courant étant le Web. Leur découverte, dans ce contexte, par des consommateurs de services, est assurée le plus souvent par des annuaires qui centralisent les offres de services.

Composition de services

La composition de services est définie dans ce manuscrit de la façon suivante :

Définition 1 (Composition de services)**.** *Un processus de mise en collaboration de services permettant d'offrir de nouveaux services dits* complexes*, ou* composites*, la base de services existant,* atomiques *ou eux-mêmes* composites*.*

La notion de service complexe ne remet pas en cause celle d'autonomie des services ; car, du point de vue de leurs clients, les services complexes exposent toujours un comportement autonome. Par conséquent, un service complexe est un service "comme les autres" : la dépendance avec les autres services requis est gérée à un niveau inférieur, invisible pour l'utilisateur final. La composition peut, par ailleurs, être effectuée selon deux approches subtilement différentes : l'*orchestration* et la *chorégraphie* de services.

- Une chorégraphie promeut le service comme élément central de la composition : elle décrit en effet le flux des messages échangés par un service Web lors de son interaction avec les autres services en présence lors de l'exécution. Il s'agit donc de gérer la composition d'une manière décentralisée puisque chaque service Web a la charge d'une partie du flot de contrôle (ou "workflow"). La spécification WSCI ("Web Services Choregraphy Interface") permet ainsi de spécifier le comportement d'un service Web par rapport au reste de la composition.
- *A contrario*, dans le cadre d'une orchestration, c'est le flot de contrôle dans sa globalité qui est mis sur le devant de la scène sous la forme d'un *processus métier*. L'orchestration dépend

d'un "orchestrateur" (ou "chef d'orchestre") responsable de la composition dans son ensemble et des échanges de message, et ce par rapport aux activités définies dans le processus.

En l'occurrence, c'est cette seconde approche d'orchestration (pour sa forte prédominance dans notre contexte industriel) qui servira de base à nos travaux d'implantation d'une composition active de services.

2.1.2 Service Web

La notion de service par elle-même est purement abstraite et ne s'encombre pas de considérations technologiques ; cependant, l'implantation la plus répandue de la notion de service est réalisée par les *services Web*. Bien qu'il n'existe pas de définition canonique de celle de service Web, nous nous reposons sur celle (suffisamment générale) proposée par Moreau [Moreau, 2009] :

Définition 2 (Service Web)**.** *Un service Web est une application autonome qui peut être utilisée et découverte par d'autres applications à travers un réseau. L'ensemble de ces mécanismes reposent sur des normes et technologies Web.*

Cette définition, bien que générique, encapsule les principes fondateurs des services Web directement hérités du concept de service : le découplage et l'autonomie du service Web ainsi constitué. Les services Web sont en effet des applications destinées à être utilisées en partie dans un cadre *machine-à-machine* de manière répartie sur le réseau. Les normes et technologies citées dans la définition sont celles que l'on retrouve usuellement dans le cadre du Web auxquelles sont ajoutées des spécificités propres aux domaines considérés.

De fait, le W3C se veut beaucoup plus spécifique d'un point de vue technologique, et introduit le langage WSDL pour la description des interfaces des services ainsi que SOAP pour la communication (le plus souvent *via* HTTP avec une sérialisation XML). On retrouve ces technologies à leurs places respectives sur la figure 2.1 qui introduit l'annuaire de services en tant que médiateur initial de la communication entre les consommateurs de services et les services eux-mêmes.

Par ailleurs, les services Web sont des composants *sans état* offrant un faible couplage. Afin d'y parvenir, des spécifications techniques pour la description et la communication ont été établies et associées aux services Web à des fins normatives.

L'émergence de spécifications propres aux services Web témoigne de l'implication des industriels dans ce nouveau paradigme, ces derniers cherchent ainsi à faire collaborer leurs grands systèmes répartis au travers du Web. Ces spécifications peuvent être divisées en deux groupes : normes de bases et normes complémentaires.

Les normes de base, qui constituent les fondations techniques des services Web, sont WSDL, UDDI et SOAP. Chacune est responsable d'un aspect fondamental du fonctionnement des services :

– WSDL ("Web Service Description Language") est un langage de type XML permettant de décrire les offres techniques et fonctionnelles des services Web [Curbera et Weerawarana, 2001, Chinnici et al., 2007]. Un document WSDL est ainsi structuré en une première partie décrivant l'interface fonctionnelle du service, et une seconde partie technique portant notamment sur les protocoles de communication à utiliser ainsi que ses points d'accès. Dans la structure d'un

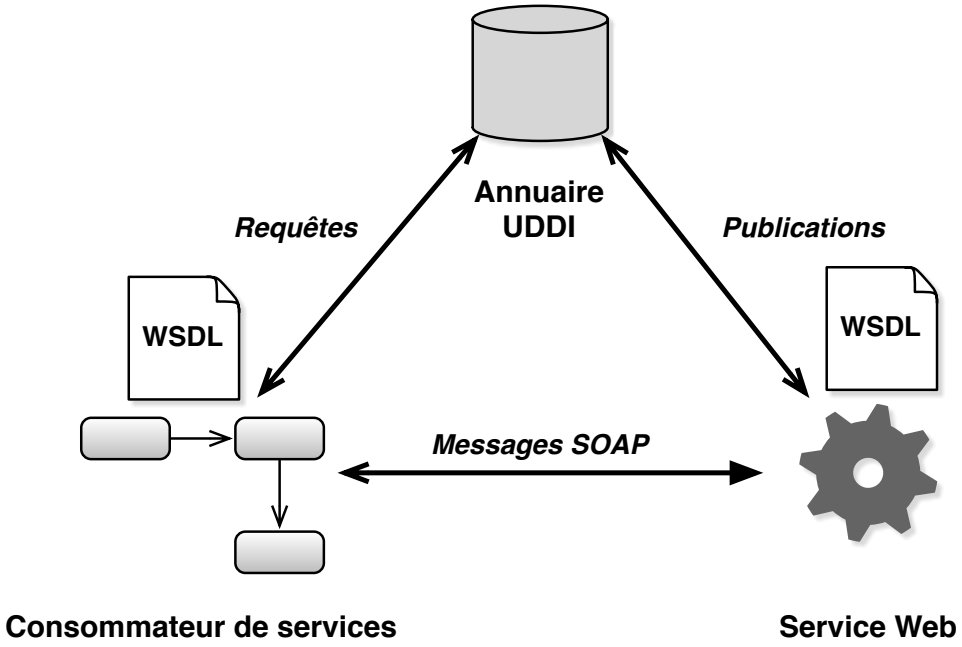

FIGURE 2.1 – Architecture des services Web selon le W3C.

document WSDL 1.1 (cf. figure 2.2), on retrouve une interface qui désigne une collection d'opérations ; un élément de liaison (*binding*) pour effectuer une association entre une interface, un protocole de transport et un format de données ; un *port* qui définit l'adresse physique d'un *binding* ; et finalement un service qui constitue une collection de *ports*.

– UDDI ("Universal Description, Discovery and Integration") est une spécification d'annuaires, mise au point par l'OASIS [5] et permettant la publication et la découverte des services, elle est cependant déployée le plus souvent dans le cadre technique des services Web. Un annuaire UDDI permet de localiser sur le réseau le service Web recherché. Il contient en outre des informations sur les fournisseurs de ces services ainsi que des méta-données sur les services (informations techniques ou légales). Un annuaire respectant la spécification UDDI est accessible au travers de plusieurs interfaces dédiées à la publication ou la recherche de services. Ainsi, un annuaire UDDI est accessible en mode *pages blanches* (la liste des entreprises fournisseuses de services ainsi que des informations associées à ces dernières), *jaunes* (les services Web de chacune des entreprises sous le standard WSDL), ou *vertes* (des informations techniques précises sur les services fournis). Grâce à cette triple lecture, l'ensemble des informations utiles sont accessibles.

– SOAP ("Simple Access Protocol") est un protocole d'échange de messages. Ces derniers sont transmis aux services Web sous forme de documents XML divisés en un *entête* (*"header"*) contenant les informations de support du message, complété d'un *corps* (*"body"*) contenant la charge utile du message, c'est-à-dire les données et opérations du domaine d'application.

5. L'OASIS est un consortium mondial qui travaille pour la standardisation de formats de fichiers ouverts fondés notamment sur XML

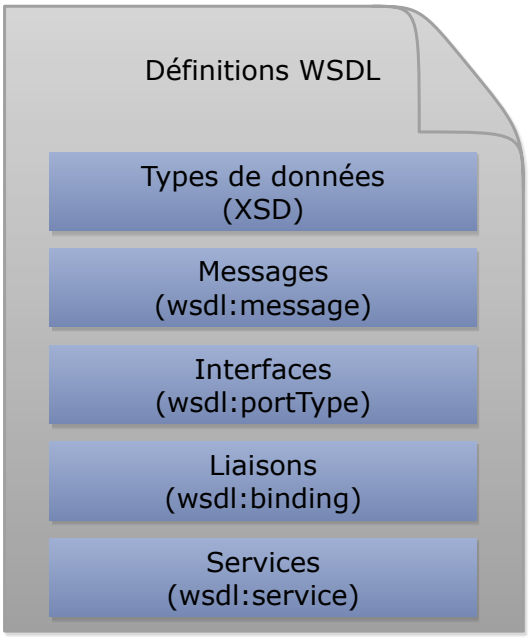

FIGURE 2.2 – Structure d'un document WSDL 1.1.

SOAP présente par ailleurs la particularité de n'être lié à aucun type de protocole de transport de données (de niveau inférieur) spécifique ; on le rencontre cependant le plus souvent associé à HTTP, ce qui exprime la composante *Web* de ces services.

Si les normes de base que nous venons d'aborder permettent d'effectuer les opérations de base du cycle de vie des services Web, elles servent de marche-pied à d'autres normes (ou spécifications) complémentaires. Il s'agit des normes communément appelées *WS-**, qui fournissent notamment des mécanismes de support de la QoS. Il est possible de les combiner, bien qu'elle soient le plus souvent à des niveaux de maturité différents car maintenues en parallèle par diverses organisations de standardisation. Ainsi, si elles peuvent effectivement se compléter efficacement, elles peuvent aussi se chevaucher, voire se concurrencer l'une l'autre. Citons, à titre d'exemple, WS-Policy qui traite spécifiquement des exigences, capacités et préférences de QoS du service, ainsi que WS-Security qui aborde les divers mécanismes permettant de mettre en place une sécurisation des échanges de messages *intra* et *extra* services.

Nous présentons ici un exemple d'offre [6] au format WSDL 1.1 d'un service Web très simple (cf. code 2.1) qui se charge uniquement d'effectuer l'affichage d'une chaîne de caractères qui lui est passée en argument *via* l'opération *print*. Cette offre de service décrit comment utiliser le service (son *API*) et quelle est son implantation concrète. S'il est possible de diviser un fichier WSDL de manière à ce qu'un service Web puisse être décrit une fois mais implanté de multiple fois, de différentes façons, nous nous restreignons dans cet exemple à la configuration de base qui dispose d'une implantation pour une description dans un fichier unique.

```
targetNamespace="http://www.eclipse.org/tptp/choreography/2004/engine/Print"
```

6. fondé sur le tutoriel suivant de le Fondation Eclipse :
http ://www.eclipse.org/stp/b2j/docs/tutorials/wsbpel/wsbpel_tut.php

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/"
                targetNamespace="http://www.eclipse.org/tptp/choreography/2004/engine/Print"
                xmlns:tns="http://www.eclipse.org/tptp/choreography/2004/engine/Print"
                xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
                xmlns:format="http://schemas.xmlsoap.org/wsdl/formatbinding/"
                xmlns:java="http://schemas.xmlsoap.org/wsdl/java/"
                >

    <!-- engine printout port -->
    <message name="PrintMessage">
        <part name="value" type="xsd:string"/>
    </message>

    <portType name="Print">
        <operation name="print">
            <input message="tns:PrintMessage"/>
        </operation>
    </portType>

    <binding name="PrintPortWsifBinding" type="tns:Print">
        <java:binding/>

        <format:typeMapping encoding="Java" style="Java">
            <format:typeMap typeName="xsd:string" formatType="java.lang.String"/>
        </format:typeMapping>

        <operation name="print">
            <java:operation methodName="print" parameterOrder="value"/>
        </operation>
    </binding>

    <service>
        <port name="JavaPrintPort" binding="tns:PrintPortWsifBinding">
            <java:address className="org.eclipse.tptp.choreography.jengine.internal.
                extensions.wsdlbinding.wsif.ports.EnginePrinterPort"/>
        </port>
    </service>

    <partnerLinkType name="printLink">
        <role name="printService" portType="tns:Print"/>
    </partnerLinkType>

</definitions>
```

Code 2.1 – Offre WSDL d'un service Web d'affichage de chaînes de caractères.

La définition d'un attribut XML *targetNamespace* permet de s'assurer que tous les éléments créés par ce fichier WSDL seront assignés à un espace de noms qui leur est propre et qui permet de les distinguer de leurs pairs. En l'occurrence les messages WSDL ainsi que le *portType* créé dans le fichier vont hériter de l'espace `"http ://www.eclipse.org/tptp/choreography/2004/engine/Print"`.

```
<!-- engine printout port -->
<message name="PrintMessage">
        <part name="value" type="xsd:string"/>
</message>
```

Les messages WSDL sont utilisés pour spécifier la forme des conteneurs de données utilisés lors de l'invocation d'une opération. Il s'agit essentiellement de listes de *parts*, chacune d'entre elles étant un type de données XSD simple ou complexe. Dans l'extrait ci-dessus, le *'PrintMessage'* est défini comme ayant une *part* nommée *'value'* de type *'xsd:string'*.

```
<portType name="Print">
        <operation name="print">
                <input message="tns:PrintMessage"/>
        </operation>
</portType>
```

Les *portTypes* représentent la définition du service Web lui-même. Ils décrivent l'*API* du service. Il s'agit d'une liste d'opérations avec des entrées (*'inputs'*) et des sorties (*'outputs'*). Chaque entrée ou sortie est un *message* WSDL qui doit avoir été défini précédemment dans le fichier WSDL ou importé. Une *operation* peut aussi disposer d'un nombre illimité d'éléments *fault* qui définissent un message d'erreur utilisé en lieu et place du message *output* par défaut. Un *portType* est donc similaire à une interface ou une classe abstraite du monde objet, il ne spécifie aucune implantation particulière, seulement ce qui peut être réalisé. On peut aussi remarquer que le *portType* (*'Print'*) n'a pas de préfixe d'espace de noms, ceci est dû au fait qu'il est automatiquement affecté au *targetNamespace* précédemment défini, comme tout élément créé au sein du fichier WSDL.

```
<binding name="PrintPortWsifBinding" type="tns:Print">
        <java:binding/>

        <format:typeMapping encoding="Java" style="Java">
                <format:typeMap typeName="xsd:string" formatType="java.lang.String"/>
        </format:typeMapping>

        <operation name="print">
                <java:operation methodName="print" parameterOrder="value"/>
        </operation>
</binding>
```

Un *binding* WSDL spécifie concrètement comment le service Web est implanté, contrairement à toutes les précédentes informations qui étaient abstraites. De fait, cette offre de service ne sera exploitable par ses clients que si elle contient un élément de ce type : un WSDL basique est une description de l'offre en termes de services qui n'est pas tenue de préciser le *où* et le *comment* technique d'accès à ce service.

Un service peut être "lié" de plusieurs façons, avec de nombreux *bindings* (cf. figure 2.3). Les plus courants sont le *binding* SOAP/HTTP qui véhicule des messages SOAP *via* le protocole HTTP lors de l'interaction avec les services Web, et le *binding* Java que nous allons exploiter dans cet exemple car il permet de définir un service Web très rapidement en liant à un *portType* une classe Java qui

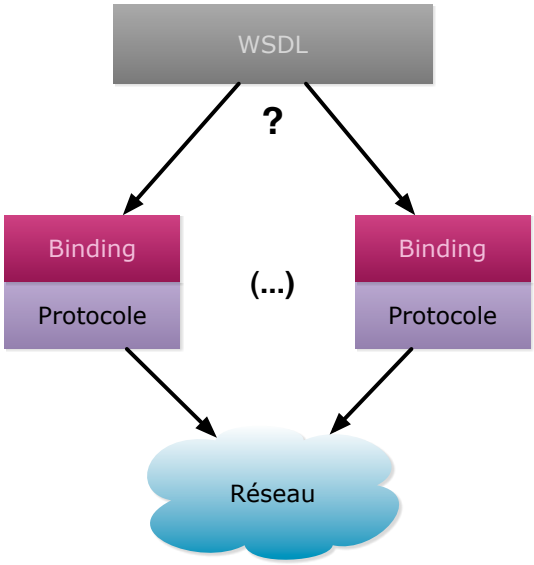

FIGURE 2.3 – De nombreux moyens d'accès à un service Web en fonction de son offre.

sera utilisée directement comme implantation de ce service. Dans le *binding* ci-dessus, l'opération *'print'* du précédent *portType* a été appariée à une méthode Java *'print'*, et le type XSD *'string'* au type Java *'String'*. A partir de cette information d'appariement, une classe Java spécifique peut être par la suite indiquée dans le fichier WSDL comme l'*address* d'une implantation de service. Cette classe sera instanciée, et lorsqu'un appel sera effectué sur la méthode *'print'* du service, elle sera transmise avec toutes les conversions de données nécessaires à la méthode Java *'print'* indiquée dans le *binding*.

```
<service>
        <port name="JavaPrintPort" binding="tns:PrintPortWsifBinding">
                <java:address className="org.eclipse.tptp.choreography.jengine.internal.
                    extensions.wsdlbinding.wsif.ports.EnginePrinterPort"/>
        </port>
</service>
```

Un élément *service* permet de définir des *ports* WSDL. Chaque *port* est une instance particulière de service Web qui est implantée *via* un *binding* spécifique et est disponible à une adresse donnée. Le *port* dans l'extrait de code ci-dessus défini un service Web lié grâce au *binding 'PrintPortWsifBinding'* et qui peut être trouvé à l'adresse *'org.eclipse...EnginePrinterPort'*. On remarque ainsi très rapidement que le type d'adresse utilisé dépend du *binding*. Le *binding* Java sait interpréter l'attribut *'className'* comme le nom qualifié d'une classe Java et comprend que cette dernière doit être instanciée et les opérations WSDL transmises aux méthodes Java indiquées dans le précédent *binding*.

Enfin, la classe Java définie par l'extrait de code 2.2 correspond à l'implantation du service telle que pointée par dans le *port*. Elle implante la méthode *'print'* telle que définie dans le *binding*.

```
package org.eclipse.tptp.choreography.jengine.internal.extensions.wsdlbinding.wsif.ports;

public class EnginePrinterPort {

        public void print(String s) {
                System.out.println(s);
        }

}
```

Code 2.2 – Implantation Java d'un service Web d'affichage de chaînes de caractères.

2.1.3 Service Web sémantique

On qualifie de *sémantique* un service Web mettant en œuvre des technologies issues de la récente rencontre entre le monde des services Web et celui du Web sémantique [Berners-Lee et al., 2001]. Ainsi, si les normes et spécification de services Web que nous avons précédemment abordées fixent un cadre syntaxique à leur utilisation, et ceci en répertoriant les fonctionnalités et les données de ces services de manière rigoureuse et réutilisable, le Web sémantique s'efforce quant à lui de les rendre "intelligible" par un programme informatique. Cette notion va ici se traduire par la capacité d'un programme à mener à bien des raisonnements complexes, de manière autonome, sur des méta-informations (ou "méta-données") venant caractériser les fonctionnalités et les données des services Web.

D'un point de vue technique, une offre de service Web (donc son interface sur le monde extérieur) est décrite en WSDL. Les informations contenues par cette description sont d'ordre technique (l'adresse du service, le protocole utilisé pour la communication, etc.) et fonctionnel (quel service est rendu par cette entité, caractérisé par ses méthodes et le type de ses données), mais vont, en tout état de cause, se présenter sous la forme d'informations syntaxiques : *la forme* des méthodes et données est décrite, mais pas *leur fond*.

Une première étape à la description du "fond" des services par rapport à leur "métier", et donc du sens accordé à leurs méthodes, données, et même aux services eux-mêmes dans leur ensemble, fut de porter des méta-informations, dans les balises de commentaires prévues à cet effet au sein des offres de services. Cependant, cette *sémantique* primitive ne permet pas l'interprétation autonome des méta-données par un programme puisque les commentaires sont, par nature, destinés à être lus par un opérateur humain : elles ne peuvent que servir de documentation ou de guide pour un ingénieur. La jonction avec les travaux du Web sémantique s'effectue alors dans une volonté commune d'élever ses méta-informations au rang de véritable sémantique métier librement interprétables par une machine : le langage naturel dans lequel sont dépeints les commentaires étant particulièrement mal adapté à un traitement automatique, ces travaux proposent alors de décrire les méta-données de manière formelle, pour certains à l'aide d'ontologies fondées sur la logique de description [Kutz et al., 2003], tel que rendu possible par le langage OWL-DL [Welty et McGuinness, 2004].

Se pose alors la question du lien entre la description syntaxique d'un service et les informations d'ordre sémantique qui le complètent. Plusieurs solutions techniques existent, avec leur avantages et inconvénients respectifs. Citons pour les plus connues OWL-S [Martin et al., 2004], WSMO [Roman et al., 2005] ou encore SAWSDL [Kopeckỳ et al., 2007]. C'est cette dernière qui sera mise à l'œuvre dans nos travaux, et ce pour des raisons aussi bien techniques qu'industrielles.

En effet, elle s'est imposée, depuis l'obtention en 2007 de son statut de "Recommandation W3C" comme *la* spécification incontournable dans le monde des services Web sémantiques. Hormis le prestige et la pérennité que lui confère un tel statut, c'est surtout sa facilité à intégrer aisément les services patrimoniaux déjà déployés dans les entreprises (SAWSDL étant défini comme extension de WSDL 2.0) qui a fini d'asseoir sa prédominance. Qui plus est, elle est aisée à déployer et ne fait pas d'hypothèses spécifiques sur le type de logique utilisé pour la description sémantique. Elle permet ainsi, par le biais des attributs *sawsdl:modelReference*, *sawsdl:loweringSchemaMapping* et *sawsdl:liftingSchemaMapping*, spécifiquement prévus par la spécification, d'annoter sémantiquement les principaux éléments syntaxiques d'une offre de service. La valeur de ces attributs étant le plus souvent une URL pointant vers un concept extrait d'une ontologie métier.

Ce principe est illustré dans l'extrait de code 2.3 extrait de la spécification SAWSDL, où un service basique de vente à distance est annoté sémantiquement par des concepts extraits d'une ontologie dont l'URL est `http://www.w3.org/2002/ws/sawsdl/spec/ontology/purchaseorder`. Ce service expose une seule méthode *order* en attente d'un numéro de compte client et d'une liste d'articles à commander, elle va ensuite retourner l'état de la commande (*confirmée*, *en attente*, *rejetée*). Chaque annotation *sawsdl:modelReference* identifie un concept bien précis de l'ontologie : l'élément *OrderRequest* du schéma de données de ce SAWSDL est ainsi décrit grâce au concept éponyme de l'ontologie *purchaseorder*. Il est possible de raisonner automatiquement sur *le concept OrderRequest*, par exemple lors d'une recherche de services dans un annuaire, car celui-ci est défini formellement dans une ontologie, par rapport aux autres concepts du domaine métier couvert par cette dernière, et ceci contrairement à *l'élément OrderRequest* qui en lui-même ne constitue qu'un nom de méthode. Un intérêt direct est alors de proposer, suite à une recherche dans un annuaire, des services sémantiquement proches ou équivalents, là où une recherche purement syntaxique (par exemple sur les noms exacts des services ou de leurs méthodes) aurait échouée.

Pour finir, une annotation *sawsdl:loweringSchemaMapping* est aussi liée à l'élément *OrderRequest*, elle pointe vers un fichier XML de description des correspondances entre les éléments internes d'*OrderRequest* et les concepts ontologiques disponibles. Cette annotation, utilisée conjointement avec *sawsdl:liftingSchemaMapping*, assoit l'ontologie comme un modèle pivot qui apporte un élément de réponse, bien que basique, aux problèmes récurrents d'adaptation de données propres aux services Web sémantiques. En effet, dans les SSOAs, à partir du moment où la mise en correspondance d'une offre et d'une demande de service est dirigée par des critères sémantiques de haut niveau extraits d'ontologies, cette correspondance ne préjuge en rien d'une parfaite correspondance, au plus bas niveau syntaxique, entre les structures de données indispensables au passage de valeurs compréhensibles par les deux parties en présence. Ainsi, par une utilisation appropriée des *lifting* et *loweringSchemaMapping*, on est capable par exemple, de mettre en correspondance deux éléments annotés par le concept ontologique commun *OrderRequest* (un élément côté processus client, l'autre côté service Web), donc équivalents sémantiquement, mais subtilement différents syntaxiquement (par exemple sous-éléments du type dans un ordre ou de noms différents, champs composites, etc.). Il suffit de décrire au préalable, sous forme de transformations XSLT, les correspondances de ces éléments vers, et à partir de, la structure du concept ontologique *OrderRequest* ; puis, dans un second temps, d'enchaîner ces transformations pour faire "naviguer" les données entre clients et services. Cette approche est synthétisée dans la figure 2.4. Bien que simple à mettre en œuvre, elle n'est cependant pas sans défaut ni inconvénient : l'approche suppose déjà l'existence de concepts ontologiques suffisamment détaillés pour contenir toutes les données à transformer, ce qui n'est pas le cas de toutes les ontologies métier ; ensuite, elle ne précise pas par quels moyens les transformations sont

```
<wsdl:description
  targetNamespace="http://www.w3.org/2002/ws/sawsdl/spec/wsdl/order#"
  xmlns="http://www.w3.org/2002/ws/sawsdl/spec/wsdl/order#"
  xmlns:wsdl="http://www.w3.org/ns/wsdl"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:sawsdl="http://www.w3.org/ns/sawsdl">

  <wsdl:types>
    <xs:schema targetNamespace="http://www.w3.org/2002/ws/sawsdl/spec/wsdl/order#"
      elementFormDefault="qualified">
      <xs:element name="OrderRequest"
          sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/
              purchaseorder#OrderRequest"
          sawsdl:loweringSchemaMapping="http://www.w3.org/2002/ws/sawsdl/spec/mapping/
              RDFOnt2Request.xml">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="customerNo" type="xs:integer" />
            <xs:element name="orderItem" type="item" minOccurs="1" maxOccurs="unbounded" />
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:complexType name="item">
        <xs:all>
          <xs:element name="UPC" type="xs:string" />
        </xs:all>
        <xs:attribute name="quantity" type="xs:integer" />
      </xs:complexType>
      <xs:element name="OrderResponse" type="confirmation" />
      <xs:simpleType name="confirmation"
          sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/
              purchaseorder#OrderConfirmation">
        <xs:restriction base="xs:string">
          <xs:enumeration value="Confirmed" />
          <xs:enumeration value="Pending" />
          <xs:enumeration value="Rejected" />
        </xs:restriction>
      </xs:simpleType>
    </xs:schema>
  </wsdl:types>

  <wsdl:interface name="Order"
      sawsdl:modelReference="http://example.org/categorization/products/electronics">
    <wsdl:operation name="order" pattern="http://www.w3.org/ns/wsdl/in-out"
        sawsdl:modelReference="http://www.w3.org/2002/ws/sawsdl/spec/ontology/purchaseorder#
            RequestPurchaseOrder">
      <wsdl:input element="OrderRequest" />
      <wsdl:output element="OrderResponse" />
    </wsdl:operation>
  </wsdl:interface>
</wsdl:description>
```

Code 2.3 – Exemple d'offre de service au format SAWSDL.

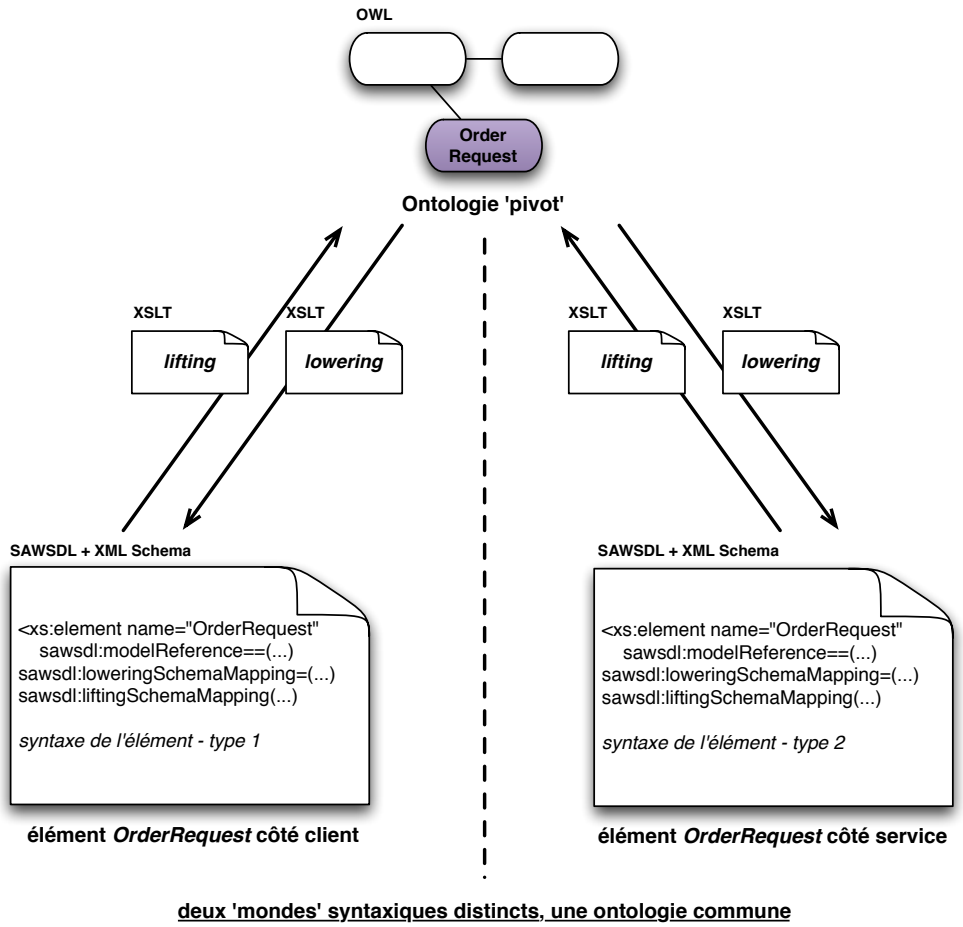

FIGURE 2.4 – Transformation de données par *lifting* et *loweringSchemaMappings*.

obtenues : génération automatique ou mise à contribution des utilisateurs ? Pour finir, elle induit la création d'un nombre très important de ces transformations vers, et à partir de, l'ontologie pivot.

Les services Web sémantiques et la technologie SAWSDL apportent donc une réponse technique non-négligeable à la problématique de la gestion d'une *grande diversité* en termes de services *équivalents* disponibles à l'exécution. Cette diversité est, par ailleurs, une caractéristique importante du cas d'utilisation de nos travaux (cf. chapitre 4). Elle est en effet favorisée par la mise en œuvre probante et effective de la propriété de couplage lâche promue par les Architectures Orientées Services, grâce à l'utilisation de sémantique métier dans les offres et demandes de services.

2.1.4 Composition de services Web

D'un point de vue technique, la spécification BPEL4WS ("Business Process Execution Language for Web Services"), appelée aussi WS-BPEL, ou plus communément BPEL, a récemment émergée comme un standard de fait, issue de la récente fusion de précédents travaux menés par IBM et Microsoft dans le domaine des langages *d'exécution* de processus métier. Le langage BPEL, que

nous étudierons plus en détail dans la section 2.2, est en effet aujourd'hui incontournable pour la spécification et la mise en œuvre de compositions de services. C'est notamment vers ce langage que se sont tournés les derniers efforts de l'industrie pour la mise au point des nombreux moteurs d'orchestration, dont les plus répandus sont abordés ci-dessous. Il faut cependant noter que, bien que fondés sur la même spécification BPEL, ils ne sont que rarement interopérables ; cet état de fait est en grande partie dû à l'implantation par certains industriels d'extensions propriétaires à la spécification originelle dans leurs moteurs.

Moteurs d'orchestration de services Web

Un *moteur d'orchestration* est une implantation du concept, précédemment évoqué, d'orchestrateur central. On ne considère ici que les moteurs capables d'interpréter la spécification de processus BPEL, les plus répandus à l'heure actuelle, et recherchons tout particulièrement le support du mécanisme d'extension défini dans WS-BPEL 2.0. Ce dernier, exploité lors de l'implantation de la liaison tardive de services (cf. 5.6), permet de définir des activités *ad hoc* dérivées des activités BPEL standards, voire de toutes nouvelles activités.

Le but de cette section n'est pas d'établir une liste exhaustive de ces moteurs ; cependant, nous nous arrêtons sur certaines caractéristiques saillantes de trois moteurs libres particulièrement répandus :

- *ActiveBPEL* d'Active Endpoints [7] est disponible en majeure partie sous la licence virale GPL v2 ce qui induit l'utilisation d'une licence similaire dans tout produit qui en serait dérivé. Il ne propose aucun mécanisme d'extension des processus métier car il implante la version 1.1 de la spécification BPEL mais dispose cependant d'une grande notoriété dans la communauté SOA, probablement due à son apparition précoce.
- *Orchestra* [8], développé par le consortium OW2, est distribué sous licence LGPL ce qui facilite son intégration ou dérivation dans des réalisations industrielles "fermées". Contrairement à ActiveBPEL, il dispose d'un véritable support des extensions d'activités spécifiés par WS-BPEL 2.0. Son usage est cependant moins répendu.
- *Apache ODE* [9] est distribué sous licence Apache v2 ce qui permet donc de fonder des produits industriels propriétaires sur sa base de code. Il dispose d'un excellent support des extensions WS-BPEL 2.0 (il fourni d'ailleurs lui-même un ensemble d'extensions pré-établies) mais reste pour l'instant relativement moins connu et répandu qu'ActiveBPEL ou Orchestra.

Autres approches de composition

Nous avons évoqué ci-dessus plusieurs alternatives pour l'orchestration de services Web, elles se fondent toutes sur l'usage d'un moteur d'orchestration qui permet une séparation stricte entre la définition d'un processus métier, client des services disponibles dans son voisinage, et les services eux-mêmes. Mais bien souvent, une approche beaucoup plus statique, bien que pragmatique, à la composition de services consiste à la décrire directement par l'usage d'un langage généraliste de

7. http ://www.activevos.com/community-open-source.php
8. http ://orchestra.ow2.org/xwiki/bin/view/Main/
9. http ://ode.apache.org/

programmation. En effet, des langages comme Java disposent de plusieurs bibliothèques permettant de manipuler des services Web. De plus, certaines approches hybrides ont fait leur apparition : c'est le cas du projet "BPEL to Java"[10] qui permet de générer à partir d'une description BPEL un programme Java capable de mener à bien l'orchestration afférente de manière autonome.

2.2 Processus métier

Cette section a pour but de définir précisément la notion de processus métier précédemment évoquée, et ce au travers de la présentation d'un ensemble non-exhaustif des solutions que l'on considère comme prédominantes sur ce segment (BPMN, diagrammes d'activités, BPEL, etc.).

La dénomination de "processus métier" est celle qui a été retenue dans ce manuscrit, mais il en existe d'autres qui recoupent un même noyau conceptuel : on peut ainsi parler de "procédures d'entreprise", "processus d'affaire", "procédés" ou, sous sa forme anglo-saxonne, de "(business) process". De toutes ces dénominations émerge une sémantique commune que nous fondons sur les travaux de Johansson *et al.* [Johansson et al., 1993] :

Définition 3 (Processus métier)**.** *Un ensemble d'activités liées qui prend une entrée et la transforme en sortie. La transformation effectuée par le procédé doit créer de la "valeur ajoutée" par rapport à l'entrée et produire donc une valeur de sortie plus "utile" au requérant du processus métier.*

Par extension de cette précédente définition, nos travaux se situent dans le cadre des processus informatisés : les entrées/sorties sont des données et les activités des instructions ou programmes externes. Ainsi, dans notre contexte, un processus métier est constitué d'un flot de contrôle, d'un flot de données (techniques et métiers), et d'un ensemble d'activités atomiques assurées en majeure partie par des partenaires externes au processus (en l'occurrence, des services).

On effectue alors la distinction entre les normes dites *de conception* de processus métier, qui se limitent à une description de haut niveau (souvent graphique) des processus, et celles *d'exécution* à proprement parler. Si elles permettent toutes de représenter de manière relativement naturelle la décomposition d'une tâche particulière en étapes distinctes, ainsi que d'exposer la logique contrôlant l'exécution de ces étapes, seuls les processus exprimés (directement ou non) par le biais d'une norme d'exécution contiennent suffisamment d'information de bas niveau pour pouvoir être exécutées de manière automatique, sans intervention humaine *a posteriori*, par un moteur d'orchestration de services (cf. 2.1.4).

2.2.1 Normes de conception

BPMN [White, 2004a] est une norme graphique, simple d'utilisation (car disposant d'un ensemble relativement réduit d'éléments ≈ 10), et adaptée à la modélisation haut-niveau de processus métiers. Les processus BPMN n'ont donc pas vocation à être directement exécutés dans un système informatique, mais sont plutôt mis en œuvre dans les phases amont de d'ingénierie logicielle, quitte à être traduits *a posteriori* (par transformation de modèle et/ou intervention humaine) dans

10. http ://www.eclipse.org/stp/b2j/

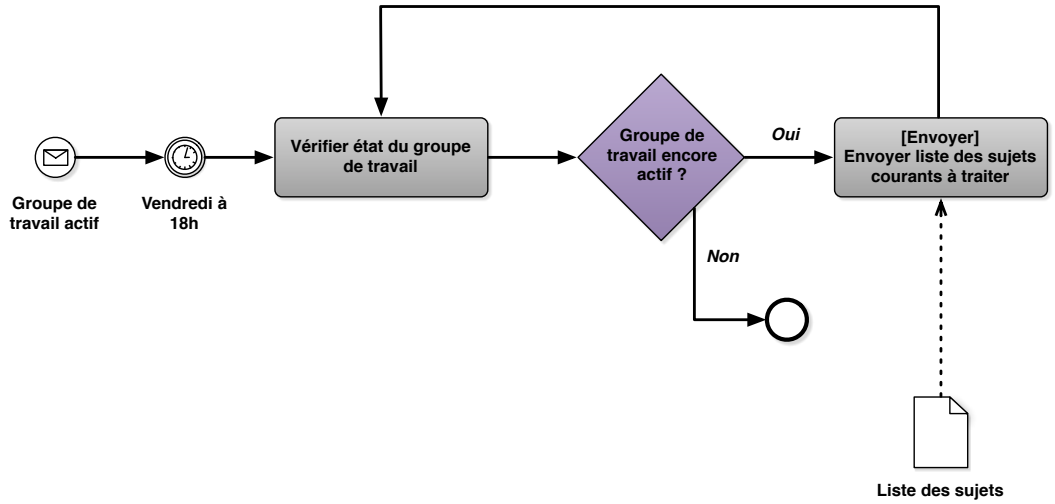

FIGURE 2.5 – Exemple caractéristique de processus BPMN.

un format exécutable. La figure 2.5 présente ainsi un exemple caractéristique de processus BPMN de haut niveau, pour la gestion d'un groupe de travail par email, où les principales activités sont effectivement décrites de manière discursive ("Vérifier état du groupe de travail", "Groupe de travail encore actif ?", etc.).

Un diagramme d'activités UML permet aussi de modéliser un processus interactif (global ou partiel) pour un système donné, et ce, qu'il soit logiciel ou non. Il est créé à partir d'une des sous-spécifications du langage de modélisation "unifié" UML [Rumbaugh et al., 2004], et une de ses forces réside justement dans sa capacité à référencer *et compléter* des modèles UML de types différents ; comme des diagrammes de classe, de cas d'utilisation, etc.

Les diagrammes d'activités peuvent alors être vus comme des représentations comportementales de haut niveau permettant d'exprimer une dimension temporelle sur tout ou partie des modèles UML considérés. En ce sens, ils sont sémantiquement proches des diagrammes UML de communication ou d'état-transition, voire des diagrammes BPMN, ce qui a déjà été étudié [White, 2004b] et peut être constaté visuellement sur la figure 2.6 représentant la simple boucle *for(A ;B ;C) D ;* d'une manière proche d'un organigramme. Ces diagrammes se distinguent cependant de BPMN dans la mesure où ce dernier est centré sur la modélisation d'un processus dans sa globalité alors que les diagrammes d'activité ont pour vocation de mettre au jour les objets-même du système.

2.2.2 Norme d'exécution : BPEL 2.0

BPEL est un langage de processus qui dispose d'une syntaxe XML et permet la mise en œuvre de services Web complexes. Ces processus sont exécutables par des moteurs d'orchestration, et constitués en majeure partie d'activités de gestion du flot contrôle et de données, ainsi que de communication avec les services Web requis et clients du processus. Dans la spécification WS-BPEL 2.0, la distinction est effectuée entre activités *basiques* et *avancées* de gestion du flot de contrôle :

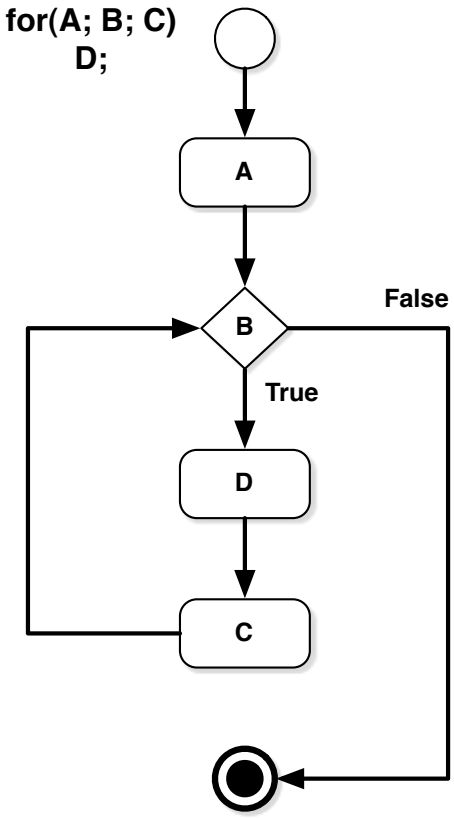

FIGURE 2.6 – Diagramme d'activité UML de la boucle *for*.

- Activités basiques : *invoke* permet d'appeler le port d'un service Web partenaire ; *receive* permet de se mettre en attente de la réception d'un message ; *reply* correspond à la réponse adressée au client du processus ; *wait* met le processus en attente pour un temps déterminé ; *assign* permet d'affecter une valeur à une variable ; *throw* lève une exception traitée ou non au sein du processus ; et finalement, *empty* qui correspond à une instruction sans effet.
- Activités avancées de gestion du flot de contrôle : *flow* exécute un ensemble d'activités de manière concurrente et reliées entre elles par des liens ; *sequence* exécute en séquence un ensemble d'activités ; *switch* permet de sélectionner une branche d'activités parmi plusieurs, en fonction de conditions ; *while* effectue des itérations jusqu'à satisfaction d'un critère, *pick* bloque le processus jusqu'à ce qu'un événement spécifique se produise (réception d'un message, alarme temporelle, etc.) ; *scope* définit une zone restreinte du processus permettant la définition de variables, de fautes et de gestionnaires d'exception ; *compensate* permet d'invoquer un processus de compensation d'exception.

Pour illustrer le plus simplement possible ces concepts, nous nous tournons vers l'inévitable exemple *'Hello World !'* bien connu des informaticiens. Mais cette fois, l'affichage de la chaîne de caractères sera effectué grâce à un service Web (cf. le service que nous avons défini dans la section 2.1.2). Le programme *'Hello World !'* est ici défini à l'aide du processus BPEL présenté dans l'extrait de code 2.4 : la définition de la logique applicative se trouve effectivement dans le code BPEL, celle des

```
<?xml version="1.0" encoding="UTF-8"?>
<process
        xmlns="http://schemas.xmlsoap.org/ws/2003/03/business-process/"
        xmlns:print="http://www.eclipse.org/tptp/choreography/2004/engine/Print"

        <import importType="http://schemas.xmlsoap.org/wsdl/"
                location="../../test_bucket/service_libraries/tptp_EnginePrinterPort.wsdl"
                namespace="http://www.eclipse.org/tptp/choreography/2004/engine/Print" />

        <partnerLinks>
                <partnerLink    name="printService"
                                partnerLinkType="print:printLink"
                                partnerRole="printService"/>
        </partnerLinks>

        <variables>
                <variable       name="hello_world"
                                messageType="print:PrintMessage" />
        </variables>

        <assign>
                <copy>
                        <from><literal>Hello World !</literal></from>
                        <to>.value</to>
                </copy>
        </assign>

        <invoke partnerLink="printService" operation="print" inputVariable="hello_world" />

</process>
```

Code 2.4 – Extrait d'un processus métier basique au format BPEL.

types des données dans des schémas XSD complémentaires, et celle des entrées/sorties du programme dans les descriptions WSDL qui lui sont associées.

```
<import importType="http://schemas.xmlsoap.org/wsdl/"
        location="../../test_bucket/service_libraries/tptp_EnginePrinterPort.wsdl"
        namespace="http://www.eclipse.org/tptp/choreography/2004/engine/Print" />
```

La déclaration *import* peut être utilisée pour importer des fichiers WSDL ou XSD externes au sein du processus BPEL. Dans les deux cas, ce mécanisme permet de décorréler du fichier BPEL des définitions tierces de types de données ou d'offres en termes de services qui peuvent alors être partagées et réutilisées par plusieurs processus. Il est à noter que les fichiers WSDL disposent d'un mécanisme similaire pour l'import de fichiers XSD ou même WSDL. Dans cet exemple, on référence *directement* un service Web défini dans le fichier WSDL *'...tptp_EnginePrinterPort.wsdl'*.

```
<partnerLinks>
        <partnerLink    name="printService"
                                partnerLinkType="print:printLink"
                                partnerRole="printService"/>
</partnerLinks>
```

Les *partnerLink* sont utilisés en BPEL pour désigner les partenaires de la composition auxquels on s'adresse. En l'occurrence il s'agit des services Web liés au processus : un *partnerLink* est indirectement apparié à la définition d'un *portType* d'une offre de service au format WSDL, donc à un service Web en particulier. Concrètement, cela signifie qu'*un couplage fort entre services et processus est établi lors de la définition de ces derniers*, ce qui rend BPEL, *en l'état*, incompatible avec les SSOAs, mais aussi avec une utilisation réellement dynamique d'annuaires.

Par ailleurs, les *partnerLinks* ne se limitent pas à désigner les entités auxquelles le processus s'adresse : ils peuvent aussi décrire comment ces entités externes s'adresse à lui. En effet, n'oublions pas qu'un processus métier, lors de son orchestration, est perçu lui aussi comme un service Web dans son environnement. Dans la définition ci-dessus, un attribut *partnerRole* caractérise le service Web auquel le processus va s'adresser ; il aurait aussi pu définir un second attribut *myRole* pointant vers la définition "externe" de ce processus sous forme d'un fichier WSDL.

```
<variables>
        <variable       name="hello_world"
                        messageType="print:PrintMessage" />
</variables>
```

Les variables sont utilisées pour contenir les données au cours de l'exécution du processus BPEL. Elles peuvent recueillir des valeurs XSD ou des messages WSDL. Dans l'exemple ci-dessus, une variable *'hello_world'* est déclarée comme contenant pour des messages WSDL du type *'print :PrintMessage'*. Ainsi, en lieu et place de l'attribut *messageType*, la variable aurait pu avoir un attribut *type* spécifiant un type XSD simple (par exemple *'xsd:string'* ou *'xsd:interger'*) ou complexe.

```
<assign>
        <copy>
                <from><literal>Hello World !</literal></from>
                <to>.value</to>
        </copy>
</assign>
```

En BPEL, les variables sont manipulées lors de l'accès aux services Web externes (*i.e.* passage d'arguments, stockage des valeurs de retours) ou par affectation directe. C'est ce dernier cas de figure qui est illustré par l'extrait de code ci-dessus : un valeur littérale de chaîne de caractères étant affectée à la variable *'hello_world'*. En l'occurrence, la variable *'hello_world'* correspond à un message WSDL disposant d'une partie *'value'* de type *'xsd:string'* que l'on référence ici par une syntaxe XPATH classique (utilisation du séparateur *'.'*).

```
<invoke partnerLink="printService" operation="print" inputVariable="hello_world" />
```

L'activité *invoke* correspond à l'invocation d'un service Web spécifique. C'est à cet emplacement que le processus BPEL transmet les données stockées dans la variable *'hello_world'* au service Web partenaire pour effectuer leur affichage. Le *partnerLink* indique indirectement au moteur d'orchestration l'"adresse" du service Web à invoquer et, *via* l'attribut *operation*, laquelle de ses méthodes doit être invoquée.

Ce que le service Web va effectivement réaliser et comment il va y parvenir n'est absolument pas spécifié dans ce processus BPEL. Une infime partie de cette information est cependant indiquée sous forme syntaxique dans l'offre de service au format WSDL que le processus référence (cf. l'offre de service d'affichage précédemment définie dans la section 2.1.2). Dans un contexte SSOA, cette information pourrait être significativement augmentée par l'utilisation de concepts ontologiques du domaine au sein de l'offre de service.

```
<partnerLinkType name="printLink">
        <role name="printService" portType="tns:Print"/>
</partnerLinkType>
```

L'élément *partnerLink* vu précédemment, référence un *partnerLinkType* qui n'est pas défini dans le fichier BPEL. Cet élément, défini dans l'extrait de code ci-dessus, doit être ajouté au fichier WSDL *'...tptp_EnginePrinterPort.wsdl'*. Cependant, les *partnerLinkTypes* ne sont pas à proprement parler des éléments de la spécification WSDL, mais bien de BPEL. C'est cette dernière spécification, postérieure à WSDL, qui requiert qu'une instance de *partnerLink* soit forcément associée à un *portType* WSDL spécifique. C'est l'élément *partnerLinkType* qui effectue cette association tout en définissant le rôle de chaque pair impliqué en son sein : en l'occurrence, dans cet exemple, seul le rôle *'printService'* du service Web doit être implanté.

Pour finir, il existe des alternatives à BPEL, malgré la sur-représentativité de ce dernier dans notre contexte : citons à titre d'exemple YAWL [Van der Aalst et Ter Hofstede, 2005], qui se présente comme un langage relativement récent de "flot de travail", fondé sur des principes issus des réseaux de Petri [Petri et Reisig, 2008] et ayant pour objectif de couvrir la majeure partie des besoins courants des modélisateurs en permettant d'intégrer et de se coupler à d'autres normes existantes. La Fondation YAWL [11] met par ailleurs à disposition l'ensemble du canevas et des outils construits autour du langage sous licence LGPL, ce qui pourrait contribuer à son succès sur le long terme.

2.3 Propriétés et contraintes non-fonctionnelles de services

Cette section a pour objet d'introduire les concepts de *propriétés* et *contraintes non-fonctionnelles* de services, ainsi que celui de *conformité* entre offres et demandes de service.

11. http ://yawlfoundation.org/

De par le caractère particulièrement concret de ces concepts, cette introduction passe en majeure partie par la présentation les travaux les plus courants s'y référant. Ces derniers permettent ainsi d'exprimer notamment, à une granularité et une précision plus ou moins élevée, les propriétés non-fonctionnelles que peuvent exhiber certains services. Au degré le plus fin, il s'agit de typer et classifier les propriétés elles-mêmes : de nombreux efforts en termes de *taxonomies* ou *ontologies* (cf. section 2.3.1) ont été pratiqués à cet effet. A une granularité moindre, se pose par ailleurs la question de l'articulation entre les propriétés non-fonctionnelles, ainsi que celle de leur évaluation par les fournisseurs et consommateurs de services : ce rôle étant généralement assuré par l'abstraction de *contrat de service* que nous détaillerons (cf. 2.3.3). Si ce manuscrit n'a cependant pas pour objet de présenter une contribution scientifique directe sur ces notions, notre approche pour la composition active de services repose sur ce contexte technique dans certaines de ses phases les plus amont, en particulier lors du filtrage des services (cf. section 5.4).

Dans notre cadre SOA, la notion de propriété non-fonctionnelle d'un service est indissociable du concept de *Qualité de Service* (*"Quality of Service"*, ou QoS). Afin de préciser cette notion, nous proposons la définition contextuelle suivante :

Définition 4 (Propriété non-fonctionnelle)**.** *Les propriétés non-fonctionnelles (ou* extra-fonctionnelles*) d'un service correspondent aux modalités de réalisation de ses fonctionnalités (ou propriétés fonctionnelles), c'est-à-dire à l'ensemble de ses caractéristiques de Qualité de Service. Les valeurs de QoS caractérisent la qualité du service telle que* rendue *à ses clients s'il s'agit de programmes informatiques, ou* perçue *par ces derniers, s'il s'agit d'opérateurs humains.*

A partir de cette première définition, il est possible de construire de manière incrémentale, celle de contrainte non-fonctionnelle.

Définition 5 (Contrainte non-fonctionnelle)**.** *Une contrainte non-fonctionnelle correspond à une exigence particulière d'un client d'un service, le plus souvent un processus métier, portant sur une propriété non-fonctionnelle offerte par ce service. Les contraintes non-fonctionnelles sont communément regroupées au sein de déclarations établies par les consommateurs de services sur la base de leur propres besoins et qui seront évalués sur la base des propriétés non-fonctionnelles des fournisseurs de services. On parle alors le plus souvent, dans ce cas, de* contrat *ou* politique *de QoS.*

Dans le contexte des SOA, la QoS devient ainsi une notion cruciale : avec la prolifération des services (Web), elle constitue un critère décisif dans le choix d'un service approprié parmi des offres de plus en plus compétitives et aux fonctionnalités souvent similaires. Dès lors elle constitue une priorité pour les fournisseurs de services et leurs partenaires, puisque chaque service possède ses propres caractéristiques de QoS, et chaque client possède des exigences spécifiques. Nous distinguons alors deux grandes catégories de Qualité de Service :

– **La QoS technique** ; il s'agit le plus souvent de propriétés non-fonctionnelles réseau telles que la *latence* ou la *disponibilité*. Les différentes taxonomies de QoS y font directement référence.

– **La QoS (du) domaine du système** ; qui permet d'exprimer des propriétés non-fonctionnelles selon des termes métiers. Leur définition est rendue possible notamment par l'utilisation d'ontologies métiers de QoS.

Ainsi, quelle que soit la spécification utilisée pour l'expression des propriétés et des contraintes non-fonctionnelles des services, la problématique la plus courante à laquelle les travaux que nous évoquons doivent faire face est celle du *calcul de la conformité* entre offres et demandes de services. On dit alors d'une offre qu'elle est conforme à une demande de service si le service décrit par cette offre peut être utilisé pour répondre à la demande initiale. La notion de conformité est donc liée à celle de *compatibilité* ; ce qui donne en logique du premier ordre :

$$\forall \alpha, \beta \ \text{Demande}(\alpha) \wedge \text{Offre}(\beta) \wedge \mathbf{Conforme}(\beta, \alpha) \implies \mathbf{Compatible}(\beta, \alpha)$$

Une offre et une demande sont alors compatibles si leur intersection est satisfiable :

$$\forall \alpha, \beta \ \text{Demande}(\alpha) \wedge \text{Offre}(\beta) \wedge \mathbf{Compatible}(\beta, \alpha) \implies \neg(\beta \wedge \alpha \subseteq \bot)$$

Sachant que les offres et demandes de services peuvent être effectuées selon un point de vue fonctionnel ou non-fonctionnel, la notion de conformité porte en elle-même les deux caractéristiques. Une offre devra donc le plus souvent être simultanément conforme fonctionnellement (elle décrit la fonctionnalité recherchée) et non-fonctionnellement (elle propose des caractéristiques de QoS acceptables dans la plage de valeurs spécifiées par la demande de service). Nous considérons alors la conformité fonctionnelle comme préalable indispensable à la conformité non-fonctionnelle : il faut s'assurer que le service est capable de fournir la fonctionnalité métier demandée avant même de considérer les propriétés de QoS qui le caractérisent.

Il faut cependant noter que la notion de conformité peut varier en fonction des besoins et applications, ainsi que des outils mis en œuvre pour en effectuer le calcul. Dans un précédent article [Châtel, 2007a], nous présentons une définition de la conformité fonctionnelle se basant sur le calcul d'héritage entre concepts ontologiques mais d'autres approches plus puissantes sont possibles tel que le calcul de *subsumption* fourni par un raisonneur sur ontologies. Li *et al.* calculent ainsi le degré de compatibilité entre offres et demandes et distinguent cinq types de relations entre concepts ontologiques (*exact*, *plugIn*, *subsume*, *intersection*, *disjoint*) [Li et Horrocks, 2004].

2.3.1 Taxonomies et ontologies de QoS

Définition 6 (Taxonomie)**.** *Une taxonomie [Cumming, 2003] est une liste structurée ou arbre, sous forme de hiérarchie avec les termes les plus généraux à son sommet. Les taxonomies furent utilisées par les biologistes pour classifier les êtres vivants en espèces, familles, etc.*

Idéalement, chaque élément (ou taxon*) dans une taxonomie devrait être mutuellement exclusif et non-ambigu. Pour chaque élément, les éléments plus précis ou plus généraux sont indiqués (s'ils existent) et les éléments liés sémantiquement peuvent aussi être indiqués. Une taxonomie se distingue des autres formes de classification telles que le dictionnaire contrôlé, la liste, le thésaurus ou les ontologies.*

Il existe aussi une variante sur ce modèle appelée taxonomie polyhiérarchique : il s'agit d'une taxonomie dont la structure est à mi-chemin entre une taxonomie classique et un thésaurus. Dans ce type de taxonomie un même élément peut apparaître à plusieurs endroits simultanément, cela est rendu possible par l'adjonction de dimensions additionnelles à la taxonomie.

Sabata *et al.* définissent une taxonomie pour la spécification de la QoS des différentes composantes (des applications aux ressources) des systèmes répartis [Sabata et al., 1997]. Pour les auteurs, la QoS est une combinaison entre métriques et politiques. Les métriques de QoS sont utilisées pour spécifier les paramètres de performance, les besoins en termes de sécurité et l'importance relative de chaque tâche du système. Dans cette taxonomie trois types de paramètres (propriétés) de performance sont définis : *timeliness*, *precision* et *accuracy*.

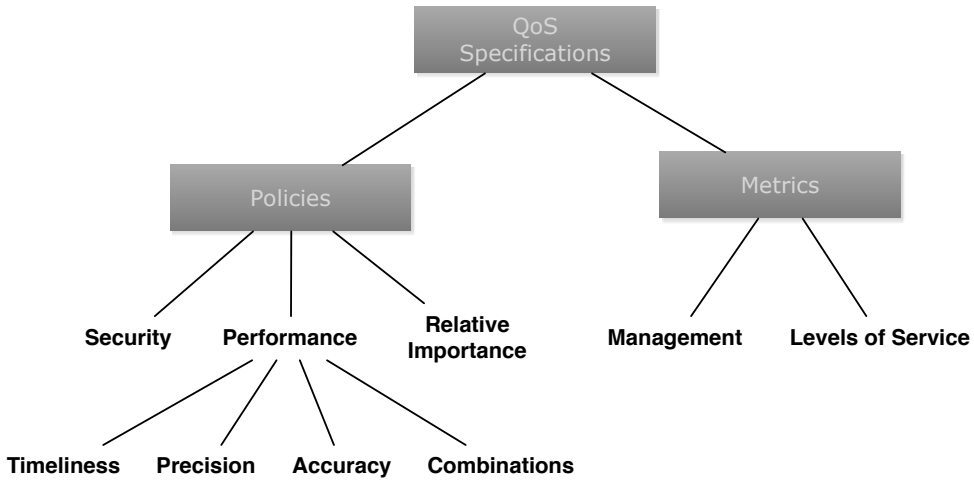

FIGURE 2.7 – Taxonomie de QoS pour Sabata *et al.*

La figure 2.7 présente la taxonomie définie par Sabata *et al.* Elle est très simple et ne propose pas de point d'extensions. Son utilisation est donc très limitée puisqu'elle n'offre pas à l'utilisateur la possibilité de définir ses propres métriques à partir des métriques de base. De plus, dans cette taxonomie les paramètres de QoS sont indissociables des métriques (il ne s'agit pourtant pas du même concept) ce qui aurait tendance à indiquer que ces travaux ne considèrent que les QoS quantitatives.

Mokhtar *et al.* présentent une classification inspirée des travaux effectués par Sabata *et al.* pour la spécification des propriétés de QoS dans les environnements AmI (*"Ambient Intelligence"*) [Ben Mokhtar et al., 2005]. Cette classification (cf. figure 2.8) est volontairement restreinte pour faciliter le calcul des correspondances entre offre et demande dans ces environnements particulièrement limités en puissance de calcul. De même que dans les précédents travaux [Sabata et al., 1997], les types de QoS sont fixés à l'avance par le canevas et ce dernier n'est pas capable de prendre en compte des nouvelles dimensions définies par l'utilisateur (il cherche à optimiser des valeurs numériques calculées à partir de formules arithmétiques définies pour chaque métrique de QoS). Cette caractéristique n'est pas limitative dans le cadre des environnements AmI mais ce canevas n'est pas suffisamment évolué pour apporter un réel gain de performance s'il était appliqué aux architectures (S)SOA.

Category	Dimension	Definition
Reliability	Availability	Probability = [0..1]
Performance	Latency	Service Time (in ms)
Cost	CPU Load	Percentage = [0..1]
	Memory	Percentage = [0..1]
	Bandwidth	Percentage = [0..1]
	Battery	Percentage = [0..1]
Security	Confidentiality	Boolean
	Integrity	Boolean
	Non-repudiation	Boolean
Transaction	Atomicity	Boolean
	Consistency	Boolean
	Isolation	Boolean
	Durability	Boolean

FIGURE 2.8 – Catégories et dimensions de QoS dans les environnements AmI.

Définition 7 (Ontologie)**.** *Une ontologie est un modèle des entités et relations dans un domaine spécifique. Elle se distingue d'une taxonomie dans la mesure où elle représente un modèle conceptuel (avec des connaissances plus complexes) voire une théorie logique (avec une représentation des connaissances très riche, complexe, consistante et significative). Une ontologie dispose d'une sémantique formelle, c'est-à-dire une théorie de modèle pour son langage. De ce fait, elle supporte l'inférence via son modèle formel, et peut être décidable et solvable en fonction de son expressivité.*

L'utilité de la définition d'une ontologie de QoS applicable à la caractérisation non-fonctionnelle des services dans une architecture SOA a été étudiée à de très nombreuses reprises : en effet, son utilisation constitue un moyen efficace pour introduire la "sémantique" dans les SOAs et les hisser au rang de *Semantic SOAs*. De fait, par la définition d'un modèle commun des connaissances du système, une ontologie de ce type permet :

- aux fournisseurs de services de publier la QoS offerte par leurs services selon une sémantique partagée par tous ;
- aux consommateurs de spécifier de manière non-ambiguë leurs besoins en termes de Qualité de Service ;
- au canevas de support des SSOAs de bénéficier des services offerts par les raisonneurs logiques sur ontologies pour faciliter le calcul de conformité entre offres et demandes non-fonctionnelles de services.

OWL-DL [McGuinness et al., 2004] est un langage de définition d'ontologies décidables particulièrement répandu dans le domaine SSOA, et dont le modèle formel est fondé sur les Logiques de Description (ou DL). Il introduit les notions de *concepts* (*i.e.* classes, entités), *propriétés* de concepts (*i.e.* attributs, rôles) et de *relations* entre concepts (*i.e.* associations). Il permet d'effectuer les inférences logiques suivantes, qui vont borner les raisonnements automatiques de certains tra-

vaux présentés dans la suite de cette section : *vérification de consistance*, *satisfiabilité des concepts*, *classification* et *réalisation*.

Cependant, le paradigme OWL s'avère inefficace pour raisonner sur des expressions numériques de QoS (inéquations, valeurs mesurées, etc.) car les services d'inférence DL ne sont pas adaptés pour traiter ce type de connaissance. Une approche multi-paradigmes semble donc nécessaire pour traiter l'ensemble des types de connaissance qui cohabitent pour décrire non-fonctionnellement les acteurs d'une architecture SOA : c'est en sens que les approches à base d'ontologies de QoS sont le plus souvent associées à d'autres technologies telles que les contrats de service (cf. section 2.3.3).

Les travaux suivants, parmi les plus connus, définissent tous des ontologies (OWL ou non) pour application dans un contexte non-fonctionnel :

WS-QoS

Tian *et al.* [Tian et al., 2003] définissent le canevas WS-QoS de support des contraintes de QoS dans les SOA ; ce canevas inclut une ontologie de QoS afin de réaliser la sélection dynamique de services Web en se fondant sur les besoins en termes de services ou de réseau. Cependant, les auteurs de cette ontologie ont choisi de l'écrire dans un pseudo-langage fondé sur une structure XML, perdant par la même occasion les avantages liés à l'utilisation de OWL (comme une sémantique bien claire et les possibilités accrues de raisonnement logique).

DAML-QoS

L'ontologie DAML-QoS [Zhou et al., 2004] a été écrite en OWL et présente certains avantages, notamment ses liens avec OWL-S utilisé pour décrire les services fonctionnellement. Initialement développée avec DAML+OIL puis OWL 1.0, un de ses principaux problèmes était la mauvaise utilisation qui était effectuée des contraintes de cardinalité OWL pour exprimer les bornes sur les propriétés de QoS. Mais ce problème a pu être levé grâce à la meilleure expressivité de OWL 1.1 et au support de la quantification sur les types des données.

Afin de permettre la définition et la mesure de la QoS dans un système, l'ontologie DAML-QoS est organisée selon un principe de couches. Elle définit trois niveaux (ou étages) distincts dans l'ontologie :

- le niveau de profil de la QoS ;
- le niveau de définition des propriétés de QoS ;
- le niveau des métriques.

QoSOnt

G. Dobson *et al.* ont travaillé sur QoSOnt [Dobson et al., 2005], une ontologie assez proche de DAML-QoS, qui se concentre sur la définition de métriques et sur le calcul de correspondance entre les besoins de QoS.

L'ontologie QoSOnt a été définie de manière suffisamment générique pour permettre son extension dans les cas particuliers, elle reste ainsi indépendante de toute vision particulière de la QoS. De plus,

elle est organisée sous forme de plusieurs couches afin de favoriser son extensibilité et sa réutilisabilité. Les couches les plus hautes sont fondées sur les couches les plus basses, on peut descendre ainsi à partir des couches domaine (par exemple *network* et *service*) jusqu'aux concepts de base de QoS. La couche la plus haute, *Usage Domains*, permet de lier les attributs de QoS définis dans la couche inférieure aux différents domaines d'application qui y sont définis. Cependant, comme pour WS-QoS, les auteurs ont défini leur ontologie dans un pseudo-langage fondé sur une structure XML.

Synthèse

Des ces différents travaux, il ressort que de nombreuses ontologies ont déjà été définies pour l'expression des notions de Qualité de Service. Les concepts qu'elles proposent sont à des niveaux d'abstractions les plus divers, cependant, de par la nature même des ontologies, il est toujours possible de dériver ses propres concepts "métiers" de classes déjà existantes. En conséquence de quoi, en cas de nouveau besoin en terme d'ontologie, il apparaît peu pertinent de développer ses propres modèles en partant de zéro, mais bien au contraire de se fonder, sinon de s'inspirer de l'existant.

Par ailleurs, il est possible de s'interroger sur le lien entre ontologies et normes (ou standards). En effet, si un standard de description des QoS (par nature généraliste) venait à s'imposer, l'utilisation d'une ontologie (permettant à tout à chacun de définir ses propres sous-concepts spécifiquement adaptés à ses propres besoins) ne saurait alors être recommandée, et ce malgré sa plus grande souplesse. A l'inverse, si une ontologie était capable de définir un ensemble de connaissances communes et fixées, pourquoi ne pas en faire alors une norme ? Si cette vision, volontairement dichotomique, peut sembler sans rapport avec la complexités des canevas logiciels faisant intervenir simultanément de multiples technologies, elle illustre cependant bien le rapport ambiguë entretenue par les deux approches, en particulier dans le monde de l'industrie qui à historiquement favorisée les approches normatives pour des raisons le plus souvent commerciales. Dans le contexte (S)SOA, de part la diversité des intervenants et la difficulté intrinsèque d'obtenir un accord entre eux, les ontologies se présentent donc, par leur souplesse, comme une solution à fort potentiel, mais qui induit la nécessité d'effectuer des calculs de correspondance sémantique.

2.3.2 Politiques

Définition 8 (Politique)**.** *Une politique est utilisée pour réguler le comportement des composants d'un système sans avoir à changer leur code ni nécessiter l'accord ou la coopération des composants impliqués. Elles guident les entités au sein d'un domaine sur leur façon d'agir en fournissant des règles sur leur comportement. En changeant de politique, un système peut être ajusté aux variations des contraintes externes qui lui sont imposées et aux conditions de l'environnement.*

De nombreuses alternatives technologiques

"Web Service Policy Framework" (WS-Policy) [Bajaj et al., 2006] est un canevas multi-usages pour la description des propriétés, besoins et caractéristiques générales des entités de l'architecture SOA. Là où WSDL permet la représentation des aspects fonctionnels d'un service, WS-Policy est utilisé pour la représentation de ses attributs non-fonctionnels. Cette spécification se présente donc

comme un complément de WSDL et non comme un remplaçant. WS-Policy définit un ensemble basique de constructions en XML qui peuvent être utilisées et étendues par d'autres spécifications de services Web pour décrire un ensemble de propriétés et besoins et servir de point de départ au calcul de correspondance. Dans ce langage, une politique est définie comme une collection d'alternatives. Chaque alternative est une collection d'assertions utilisées pour représenter un besoin, une propriété ou un comportement d'un service Web et peut avoir un nombre indéfini d'attributs et de fils.

SemPolicy [Verma et al., 2005] est une spécification de représentation des contraintes et des propriétés non-fonctionnelles des services Web qui est une extension de WS-Policy intégrant des expressions sémantiques extraites d'ontologies non-fonctionnelles. Elle se fonde sur le postulat que le manque de sémantique dans les politiques diminue l'efficacité du calcul de compatibilité entre les politiques, et propose alors l'utilisation conjointe d'ontologies et de règles SWRL pour effectuer ce calcul. Une ontologie OWL de WS-Policy a été mise au point (sorte de méta-modèle de WS-Policy) ainsi chaque politique peut être chargée dans le système comme une instance de cette ontologie.

On retrouve par ailleurs cette idée d'utiliser des ontologies dans les travaux de Uszok *et al.* sur la gestion des contrats et des politiques pour les services Web sémantiques [Uszok et al., 2004, Tonti et al., 2003]. KaoS se présente comme un système complet pour la spécification, la gestion, l'analyse et l'application de politiques représentées en OWL. Il provient de recherches effectuées initialement dans le cadre des applications à agents logiciels et du "grid computing".

Une politique KAoS est une déclaration contraignant l'exécution d'un type particulier d'action par un ou plusieurs acteurs en fonction de différents aspects d'une situation donnée : une action est définie comme une classe ontologique utilisée pour clarifier les instances d'actions qui sont souhaitées ou en cours d'exécution. KaOS utilise par ailleurs des concepts issus d'ontologies (encodées en OWL) pour construire les politiques.

Suivant une philosophie générale similaire, le langage Rei [Kagal, 2002] de spécification de politique est fondé sur une combinaison de OWL-Lite, variables logiques et de règles. L'expression de relations est possible car Rei est basé sur la Logique du 1er Ordre qui apporte cette capacité par rapport à la Logique Propositionnelle. Le langage Rei définit ainsi plusieurs ontologies basiques indépendantes de tout domaine mais va aussi nécessiter des ontologies métiers, définissant des concepts liés au domaine d'application de chaque politique. Les ontologies basiques sont, tout comme pour KAoS, extensibles par le modélisateur.

Synthèse

Les politiques mettent en exergue l'importance d'une régulation du comportement des systèmes en fonction de multiples contraintes, en particulier non-fonctionnelles. Les travaux effectués sur le sujet sont là aussi particulièrement riches et nombreux. Cependant, ils s'inscrivent le plus souvent dans une approche relativement statique de l'adaptation du comportement des composants de ces systèmes, par rapport aux conditions de l'environnement. Dans un contexte SSOA où les composants sont effectivement des services, les variations ne correspondent toutefois pas à de grandes tendances environnementales répertoriées desquelles ils est aisément possible de pré-établir des politiques, mais à une multitudes de variations et changements locaux en terme de QoS de chacun de ces services, ainsi que de leur apparition et disparition fréquente au court du cycle de vie d'un même processus métier (en particulier pour le type de processus métiers mis à l'œuvre dans notre cas d'utilisation au

chapitre 4). Ces contraintes spécifiques nécessitent alors la mise en place d'une approche adaptée, particulièrement dynamique.

2.3.3 Contrats de Qualité de Service

Il est possible de voir les contrats comme un regroupement d'assertions sur les propriétés d'un programme. Chacune d'elle constitue alors une expression booléenne à satisfaire pour son exécution [Floyd, 1967, Hoare, 1969]. Pour la suite de ces travaux, nous proposons cette vision communément admise des contrats :

Définition 9 (Contrat de Qualité de Service)**.** *Dans un contrat de Qualité de Service (ou SLA, "Service Level Agreement"), les contraintes non-fonctionnelles sont rattachées aux entités qu'elles caractérisent via des préconditions, effets, post-conditions ou invariants dans le cadre de la modélisation d'un accord le plus souvent bilatéral entre un consommateur et un producteur de service.*

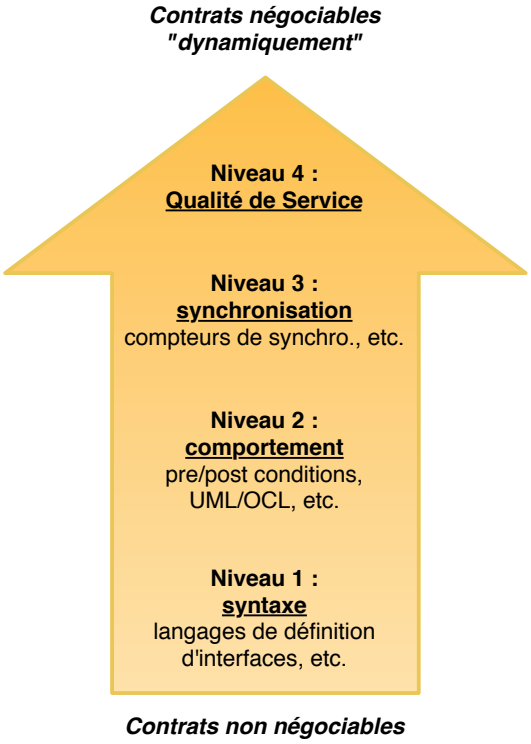

FIGURE 2.9 – Les 4 niveaux de contrats selon Beugnard *et al.*

Des travaux reconnus de l'ENST/Irisa [Beugnard et al., 1999] établissent une classification en quatre niveaux (cf. figure 2.9) des différentes formes de contrat pour les architectures à base de composants. Dans cette classification, les contrats sont rangés par degré de "négociabilité" : en partant des contrats non-négociables jusqu'aux contrats négociables à l'exécution du système.

Dans cette section, nous nous intéressons tout particulièrement aux contrats de Qualité de Service (niveau IV) dans les architectures SOA, mais nous aborderons aussi ceux applicables aux architectures à base de composants logiciels qui pourraient éventuellement être adaptés aux SOAs. En effet, contrairement aux spécifications bien établies dans le domaine fonctionnel des services Web (par exemple WSDL, SOAP ou UDDI), il n'existe pas de standards officiellement reconnus comme tels en ce qui concerne la description et l'établissement des contraintes de QoS.

GCCL

Malenfant *et al.* ont mis au point le langage de contrat GCCL ("Generalized Common Contract Language") pour répondre à certaines problématiques liées à l'assemblage des composant et au calcul de leur correspondance fonctionnelle et non fonctionnelle par rapport aux contrats établis. [Malenfant et al., 2002]. Ce travaux permettent en fait une adaptation de la programmation contractuelle au domaine du Génie Logiciel des composants et on pour objectif de fournir le moyen :

- d'exprimer des besoins fonctionnels et non fonctionnels ainsi que définir leur sémantique. Pour ce faire, il fait principalement usage de *pré*, *post* conditions et d'invariants ;
- de certifier la justesse d'une composition d'un ensemble de composants ;
- de vérifier, à l'exécution, que les composants et le canevas honorent leurs obligations contractuelles.

De QML à CQML+

QML ("QoS Modeling Language") [Frølund et Koistinen, 1998b] permet la spécification de contraintes non-fonctionnelles. Ce langage présente une approche contractuelle basée sur les trois entités suivantes : les *types de contrats*, les *contrats* et les *profils*. En QML, les aspects non-fonctionnels sont décrits en spécifiant les niveaux attendus de qualité : on cherche ainsi à déterminer si une valeur est préférable à une autre de manière à inférer les compatibilités entre contrats. La vérification de compatibilité des contraintes non-fonctionnelles d'une architecture peut ainsi être effectuée statiquement, avant exécution. Cette approche statique limite directement l'adaptabilité de ces architectures à l'exécution.

Par la suite, CQML [Aagedal, 2001], suivi de CQML+ [Rottger et Zschaler, 2003] vont successivement bâtir sur les fondations posées par QML. Ces travaux vont respectivement introduire la notion de *catégorie* comme brique de base de la spécification de QoS, puis celle d'un véritable *méta-modèle d'exécution* dans le cadre duquel les contrats peuvent être définies. Il s'agit là d'un premier pas vers une approche plus dynamique pour leur vérification.

WS-Agreement

La spécification WS-Agreement définit un langage de syntaxe XML, ainsi qu'un un protocole pour capturer la relation entre un consommateur et un producteur de service [Andrieux et al., 2004] : chaque SLA va définir un ou plusieurs SLO ("Service Level Objective") qui indiquent les besoins et les compétences de tout participant à l'accord de service, relativement à la disponibilité de ses ressources et la qualité de son service.

Dans le canevas WS-Agreement, un contrat est le résultat d'une première étape de négociation entre producteur et consommateur de service. Cette négociation est rendue possible par l'offre par un fournisseur de plusieurs niveaux non-fonctionnels de service (appelés templates) au sein d'une même spécification : par exemple, un producteur de service pourra indiquer qu'il offre différents niveaux possibles de QoS dans son contrat ; à la charge du consommateur de choisir une offre en fonction de ces propres capacités et des contraintes qu'il devra respecter par rapport au producteur.

Synthèse

Les évolutions technologiques récentes en terme de contrats (par exemple WS-Agreement) sont souvent liées à celles des SOAs. De ce fait, elle se révèlent comme un support intéressant pour la spécification de contraintes non-fonctionnelles de services. Cependant, une approche de composition *uniquement* fondée sur les contrats ne saurait répondre à toutes les facettes de la problématique que nous avons levée dans l'introduction de ce manuscrit, pour notre contexte : en effet, les contrats sont particulièrement focalisés sur les étapes de définition et de négociation des besoins entre consommateurs et fournisseurs de services, mais restent habituellement muets quant aux mécanismes à mettre en œuvre, à l'exécution, pour traiter les évolutions en terme de Qualité de Service, quand bien même celles si seraient cantonnés aux bornes acceptables décrites dans lesdits contrats.

2.4 Conclusion

Dans ce premier chapitre de contexte, nous avons effectué un panorama des différentes solutions techniques déjà à notre disposition pour l'implantation des solutions à la problématique détourée lors de l'introduction de ce manuscrit. Sans être exhaustif, il nous permet de poser les éléments (qui nous seront utiles par la suite) de service Web, de SSOA ainsi que de contrats de Qualité de Service. Ce chapitre met aussi en avant les absences auxquelles nos contributions devront répondre, notamment en terme de coordination agile (c'est-à-dire active et utile) de services. Ce sentiment devrait être conforté par l'état de l'art (cf. section 3.3), plus spécifique, que nous effectuons au chapitre suivant sur la composition dynamique de services sous contraintes non-fonctionnelles. Pour finir, si nous avons été ici amenés à balayer un champ technologique plus large que le sous-ensemble qui sera effectivement utilisé dans le cadre général de nos travaux, cette approche a été motivée par la nécessité d'assurer une compatibilité la plus large possible à l'implantation.

Chapitre 3

Etat de l'Art

Sommaire

Ce chapitre introduit les travaux connexes aux contributions scientifiques développées dans cette thèse. Il s'agit de contributions majeures représentant l'état de l'art dans les domaines du *raisonnement fondé sur une approche linguistique* (cf. section 3.1), de la *modélisation compacte de préférence* (cf. section 3.2) et de la *composition dynamique de services* sous contraintes non-fonctionnelles (cf. section 3.3).

3.1 De l'approche linguistique pour raisonner

Tel que nous l'avons précédemment évoqué dans l'introduction de ce manuscrit, un élément que nous avançons en réponse à la problématique de mise en œuvre d'une composition agile de services consiste à mettre au point un modèle qualitatif de préférences utilisateur. Ces préférences seront employées, lors d'un processus de *décision multi-critères*, pour maximiser l'utilité des liaisons entre producteurs et consommateurs de services.

Plusieurs approches sont envisageables pour effectuer cette décision [Gonzales et Perny, 2005, Boutilier et al., 2004, Brafman et Domshlak, 2002, Herrera et al., 2009], dont certaines qui utilisent conjointement des notions issues de la logique floue et du calcul avec les mots ("Computing with Words" ou CW), notions assez proches l'une de l'autre [Zadeh, 1996].

Etant donné que nous avons vocation à gérer l'*imprécision des perceptions utilisateur* lors de l'élicitation des préférences qualitatives, il semble nécessaire d'introduire en premier lieu, dans cette section, les concepts théoriques et computationnels qui sous-tendent la prise en compte de ces aspects qualitatifs et subjectifs par une approche linguistique. Nous détaillerons ainsi en 3.1.1 certains concepts propres à une modélisation floue des concepts, avant de nous attarder sur le raisonnement qu'il est possible d'y appliquer dans la sous-section 3.1.2, puis, en 3.1.3, nous établirons une liste non-exhaustive des formalismes se fondant sur la notion de 2-tuple.

3.1.1 Modélisation floue des concepts

L'approche linguistique représente les aspects qualitatifs sous forme de valeurs exprimées, par le biais de variables linguistiques [Zadeh, 1975]. Des descripteurs linguistiques appropriés doivent être choisis pour former l'ensemble des termes ainsi que leur sémantique. L'univers du discours sur lequel l'ensemble des termes est défini peut être arbitraire mais est, en pratique, souvent ramené à $[0, p]$ où $p = 1$. Par ailleurs, les ensembles de termes à cardinalité impaire, généralement 5, 7 ou 9, sont préférés [Delgado et al., 1993, Herrera et Martínez, 2000], le terme médian est alors évalué à "approximativement 0.5" et les autres termes placés symétriquement autour de lui.

Par exemple, un ensemble de 5 termes T pourrait être défini tel que : $T = \{s_0 : \textit{very low}, s_1 : \textit{low}, s_2 : \textit{medium}, s_3 : \textit{high}, s_4 : \textit{very high}\}$.

Selon un modèle de calcul linguistique purement symbolique [Yager, 1981], il est aussi requis que soient définis sur cet ensemble l'opérateur de négation *Neg*, et les opérateurs *max* et *min* [Herrera et Martínez, 2000] de telle sorte que :

1. $\text{Neg}(s_i) = s_j$ tel que $j = g - i$ (où $g + 1$ est la cardinalité),
2. $\max(s_i, s_j) = s_i$ if $s_i \geq s_j$,
3. $\min(s_i, s_j) = s_i$ if $s_i \leq s_j$.

Cependant, dans le cadre d'une modélisation floue des concepts, la sémantique des termes est donnée par leurs fonctions d'appartenance, selon la *théorie des sous-ensembles flous*. Ainsi, des fonctions trapézoïdales linéaires par morceaux (et même triangulaires) sont le plus souvent considérées comme suffisantes pour capturer l'imprécision des perceptions utilisateur.

Nous reproduisons ci-dessous certains concepts fondateurs de cette théorie de référence, ainsi que les notations que nous allons utiliser, tous issus de l'ouvrage de référence de Bouchon-Meunier [Bouchon-Meunier, 1995].

- Soit $F(X)$ l'ensemble de tous les sous-ensembles flous (SEF) de X (ensemble de référence, encore appelé domaine ou univers).
- Soit A un sous-ensemble flou de X, appartenant à $F(X)$. A est caractérisé par sa fonction d'appartenance notée $f_A(x)$, pour tout $x \in X$, c'est-à-dire $f_A : X \rightarrow [0, 1]$. La figure 3.1 montre un exemple de fonction d'appartenance de forme quelconque.

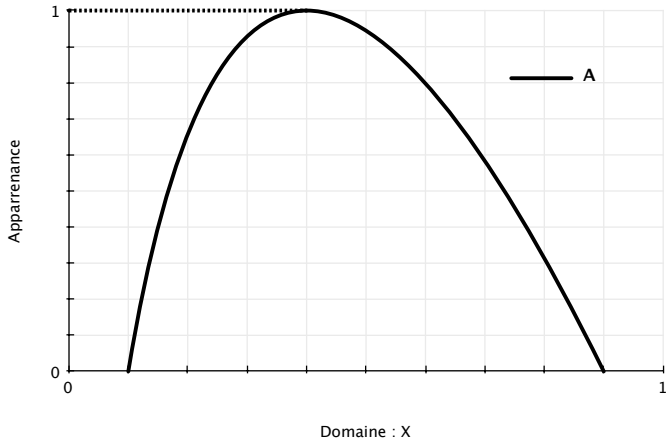

FIGURE 3.1 – Exemple de fonction d'appartenance du sous-ensemble flou A.

– La hauteur, notée $h(A)$, du sous-ensemble flou A de X est la plus grande valeur prise par sa fonction d'appartenance :

$$h(A) = \sup_{x \in X} f_A(x)$$

Dans l'exemple de la figure 3.1, $h(A)$ vaut 1.

– Un sous-ensemble flou est dit *normalisé* si sa hauteur vaut 1.

– On note $\text{Noy}(A)$ le noyau de A qui correspond à toutes les valeurs x de X pour lesquelles $f_A(x) = 1$.

– On note $\text{Supp}(A)$ le support de A qui correspond à toutes les valeurs x de X pour lesquelles $f_A(x) \neq 0$.

– Pour représenter le sous-ensemble flou $A \in F(X)$, on note

$$\begin{array}{ll} A = \sum_{x \in X} f_A(x)/x, & \text{si } X \text{ est dénombrable} \\ A = \int_X f_A(x)/x, & \text{sinon} \end{array}$$

– On appelle *intervalle flou* un sous-ensemble flou convexe, normalisé. Dans la figure 3.1, A est un intervalle flou.

– On appelle *nombre flou* un intervalle flou dont le noyau est réduit à un point.

– On appelle *intervalle flou de type L-R* ou sous-ensemble flou trapézoïdal un intervalle flou dont la fonction d'appartenance est définie entièrement grâce à des droites.
Plus précisément, soient quatre paramètres réels (m, m', a, b), a et b étant strictement positifs, et deux fonctions, notées L et R (pour Left et Right), définies sur l'ensemble des réels positifs, à valeurs dans $[0, 1]$, semi-continues supérieurement, telles que $L(0) = R(0) = 1$, $L(1) = 0$ ou $\forall x \in]0, 1[,\ L(x) > 0$, avec $\lim_{x \to \infty} L(x) = 0, R(1) = 0$ ou $\forall x \in]0, 1[, R(x) > 0$ avec

$\lim_{x\to\infty} R(x) = 0$. Un intervalle flou est de type L-R si sa fonction d'appartenance est définie par :

$$f(x) = \begin{cases} L((m-x)/a) & \text{si } x \leq m \\ 1 & \text{si } m < x < m' \\ R((x-m')/b) & \text{si } x \geq m' \end{cases}$$

- On appelle *nombre flou de type* L-R ou sous-ensemble flou triangulaire un intervalle flou de type L-R dont le noyau est réduit à un point, c'est-à-dire, pour lequel m et m' sont confondus.
- Soient A et B deux sous-ensembles flous de X. L'*intersection* de A et B est le sous-ensemble flou C, noté $A \cap B$, tel que :

$$\forall x \in X, f_C(x) = \min(f_A(x), f_B(x))$$

- Soient A et B deux sous-ensembles flous de X. L'*union* de A et B est le sous-ensemble flou D, noté $A \cup B$, tel que :

$$\forall x \in X, f_D(x) = \max(f_A(x), f_B(x))$$

- Pour l'intersection et l'union, les opérateurs choisis par défaut sont min et max, mais d'autres sont possibles : plus généralement, il peut s'agir en fait d'une norme triangulaire (*t-norme*, notée $\top$) et d'une conorme triangulaire (*t-conorme*, notée $\perp$) respectivement.
- Une t-norme est une fonction $\top : [0,1] \times [0,1] \to [0,1]$ telle que $\forall x, y, z \in [0,1]$:
  - $\top(x,y) = \top(y,x)$ (commutativité)
  - $\top(x, \top(y,z)) = \top(\top(x,y), z)$ (associativité)
  - $\top(x,y) \leq \top(z,t)$ si $x \leq z$ et $y \leq t$ (monotonie)
  - $\top(x,1) = x$ (1 est élément neutre)

  Les fonctions suivantes respectent ces propriétés : $\min(x,y)$, $x.y$ et $\max(x+y-1, 0)$.
- Une t-cornorme est une fonction $\perp : [0,1] \times [0,1] \to [0,1]$ telle que $\forall x, y, z \in [0,1]$:
  - $\perp(x,y) = \perp(y,x)$ (commutativité)
  - $\perp(x, \perp(y,z)) = \perp(\perp(x,y), z)$ (associativité)
  - $\perp(x,y) \leq \perp(z,t)$ si $x \leq z$ et $y \leq t$ (monotonie)
  - $\perp(x,0) = x$ (0 est élément neutre)

  Les fonctions suivantes respectent ces propriétés : $\max(x,y)$, $x+y-x.y$ et $\min(x+y, 1)$.
- Une *relation floue* $\mathcal{R}$ entre r ensembles de référence $X_1, X_2, \ldots X_r$ est un sous-ensemble flou de $X_1 \times X_2 \ldots \times X_r$, de fonction d'appartenance $f_{\mathcal{R}}$.
- Si $X = Y$, une relation floue $\mathcal{R}$ définie sur les deux univers X et Y est une *relation binaire floue* définie sur X.
- La composition de deux relations floues $\mathcal{R}_1$ sur $X \times Y$ et $\mathcal{R}_2$ sur $Y \times Z$ définit une relation floue $\mathcal{R} = \mathcal{R}_1 \circ \mathcal{R}_2$ sur $X \times Z$ de fonction d'appartenance définie par :

$$\forall (x,z) \in X \times Z, f_{\mathcal{R}}(x,z) = \sup_{y \in Y} \min(f_{\mathcal{R}_1}(x,y), f_{\mathcal{R}_2}(y,z))$$

- Une *variable linguistique* est représentée par un triplet (V, X, T_V) tel que V est le nom de la variable, X est l'univers des valeurs prises par V, $T_V = \{A_1, A_2, ...\}$ est l'ensemble des SEF de X utilisés pour caractériser V. Par exemple (Bande-Passante, $[0,1]$, {Faible, Moyenne, Haute}).

- Un *modificateur linguistique* est un opérateur *mod* qui permet de passer d'un SEF A à un autre SEF $mod(A)$ dont la fonction d'appartenance est $f_{mod(A)} = t_{mod}(f_A)$ avec t une transformation mathématique.

- On appelle *proposition floue élémentaire* la qualification "V est A" d'une variable linguistique (V, X, T_V), où A est un SEF de T_V ou de $mod(T_V)$, avec *mod* un modificateur linguistique de T_V. La valeur de vérité p_A de "V est A" est la fonction d'appartenance de A f_A.

- On appelle *proposition floue générale* la composition de propositions floues élémentaires de variables linguistiques qui peuvent être distinctes. Sa valeur de vérité p correspond à l'agrégation des valeurs de vérité p_A et p_B de chaque proposition floue élémentaire.
  - Exemple 1 : "V est A et W est B" : $p_{A \wedge B} = \min(p_A, p_B)$
  - Exemple 2 : "V est A ou W est B" : $p_{A \vee B} = \max(p_A, p_B)$

- $p \implies q$ équivaut, en logique classique, à $\neg p \vee q$. En logique floue, il n'y a pas une seule définition de l'implication. Par exemple, l'extension de la définition précédente est appelée *l'implication de Kleene-Dienes* et équivaut à $\max(1 - f_A(x), f_B(y))$.
  - L'implication "V est $A \implies W$ est B" entre deux propositions floues se lit "si V est A alors W est B".
  - L'implication entre deux propositions floues est elle-même une proposition floue dont la valeur de vérité est donnée, pour une fonction Φ de $[0,1] \times [0,1] \to [0,1]$, par la fonction d'appartenance $f_{\mathcal{R}}$ d'une relation floue $\mathcal{R}$ entre X et Y telle que :

$$\forall x \in X, \forall y \in Y, f_{\mathcal{R}}(x, y) = \Phi(f_A(x), f_B(y))$$

  - L'implication décrit le lien causal entre "V est A" et "W et B".

- Un *partitionnement flou* consiste à définir n SEF F_i de façon à recouvrir X. C'est-à-dire que pour tout x de X, il faut assurer une appartenance minimale ϵ à l'union des SEF :

$$\forall x \in X, f_{F_1}(x) \vee ... \vee f_{F_i}(x) \vee ... \vee f_{F_n}(x) \geq \varepsilon$$

  La figure 3.2 expose, à titre d'exemple, un partitionnement flou de la température en trois SEF distincts : *Froid*, *Tempéré*, *Chaud*.

- On appelle *fuzzification* le processus qui consiste à attribuer à la valeur réelle d'une entrée donnée sa fonction d'appartenance à chacune des classes préalablement définies, par exemple, par un partitionnement flou. On effectue donc une transformation de l'entrée réelle en un SEF. La fuzzification consiste à associer à une mesure de la variable x_0 une fonction d'appartenance par le biais d'un opérateur de fuzzification. Son choix est conditionné par la nature de la mesure : si elle est exacte (précise), on obtient un SEF singleton (cf. figure 3.3) ; sinon, le SEF peut être de forme triangulaire $f(x) = \max(0, 1 - \frac{|x - x_0|}{\epsilon})$ (cf. figure 3.4), trapézoïdale (cf. figure 3.5), etc.

- Soit A un SEF, on appelle *défuzzification* le passage inverse de A à une valeur réelle. On considère alors, à titre d'exemple, A comme représenté par la figure 3.6, et illustre quatre méthodes de défuzzification :

  1. Principe du maximum (cf. figure 3.7) :

$$D(A) = \frac{x_0}{f_x(x_0)} = \sup(f_A(x))$$

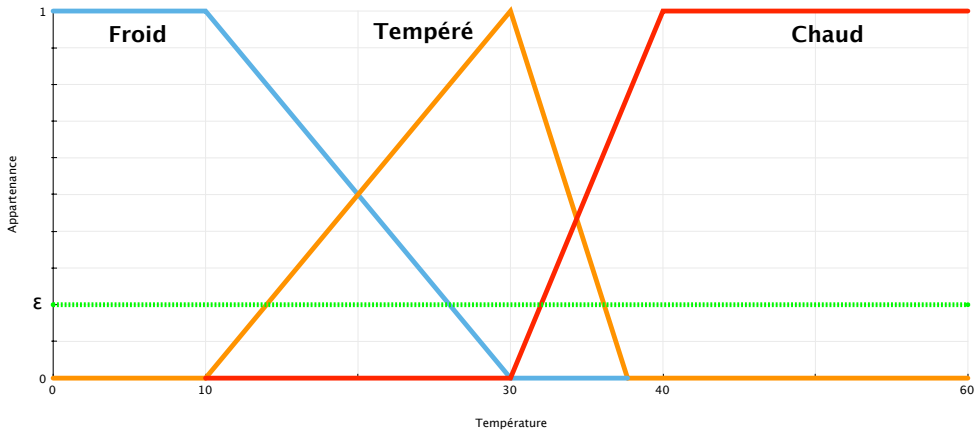

FIGURE 3.2 – Un partitionnement flou de domaine de la température.

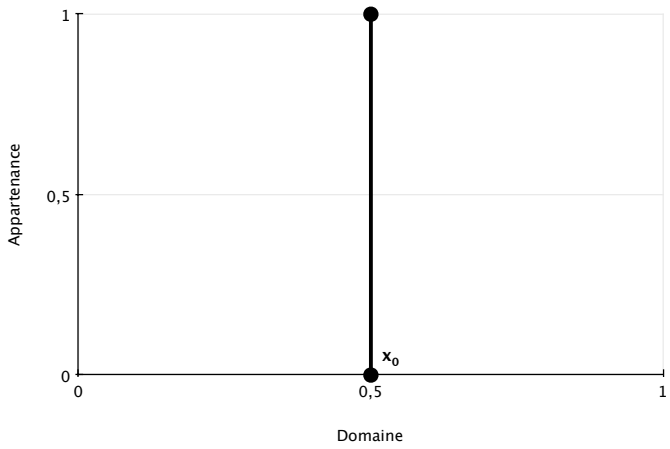

FIGURE 3.3 – Fuzzification par SEF singleton.

2. Moyenne des maxima (cf. figure 3.8).
3. Egalité des intégrales (cf. figure 3.9) :

$$D(A) = \frac{x_0}{\int_a^{x_0} f_A(x)dx} = \int_{x_0}^{b} f_A(x)dx$$

4. Calcul du barycentre pour un univers X (cf. figure 3.10) :

$$D(A) = x_0 = \frac{\int_X x.f_A(x)dx}{\int_X f_A(x)dx}$$

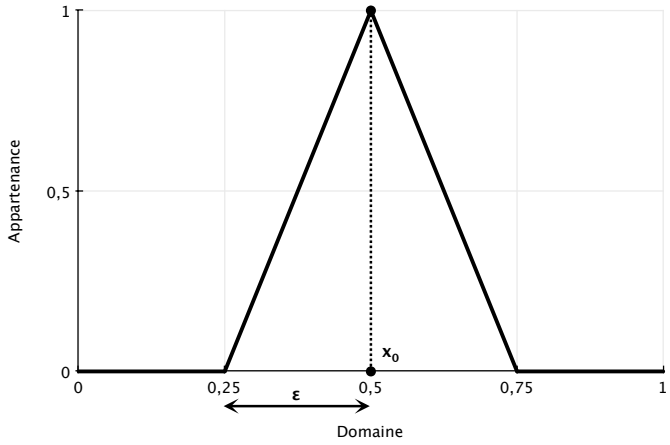

FIGURE 3.4 – Fuzzification par SEF triangulaire.

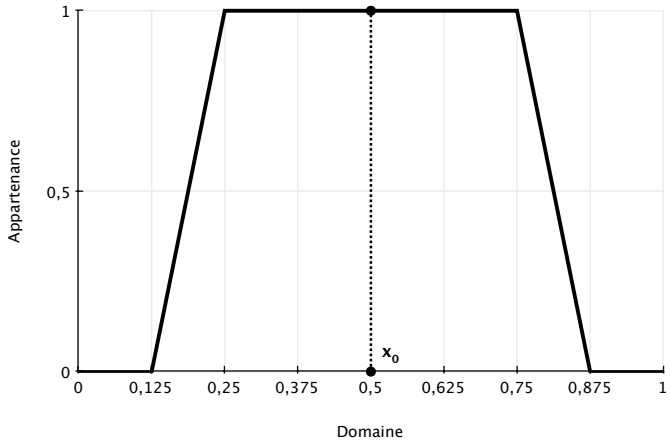

FIGURE 3.5 – Fuzzification par SEF trapézoïdal.

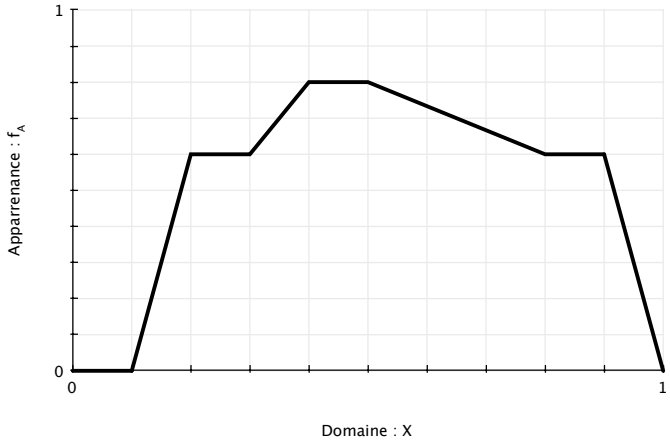

FIGURE 3.6 – Un SEF A à défuzzifier.

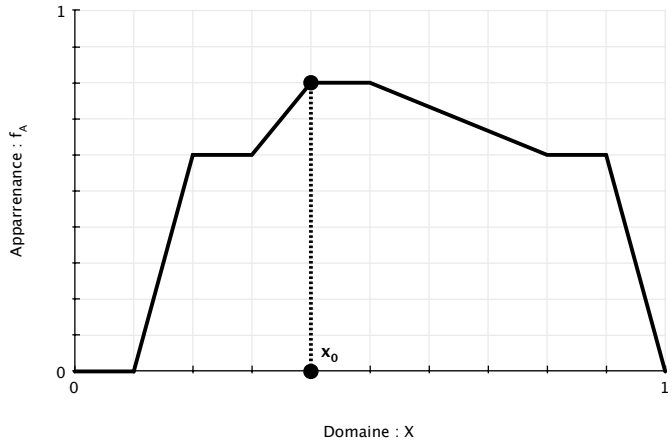

FIGURE 3.7 – Défuzzification de A : principe du maximum.

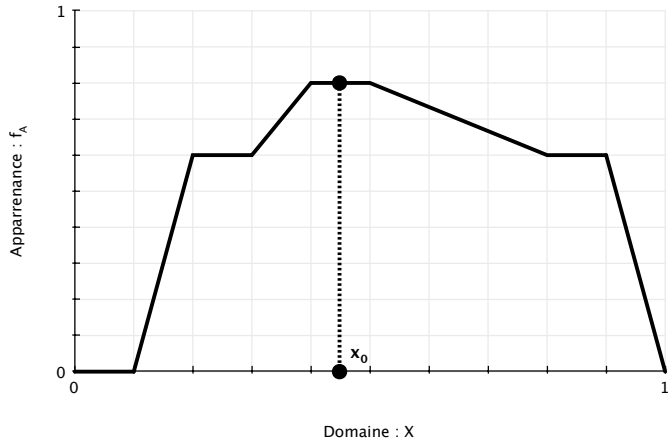

FIGURE 3.8 – Défuzzification de A : moyenne des maxima.

3.1.2 Raisonnement flou

Toujours en nous fondant sur les notions avancées par l'ouvrage de référence de Bouchon-Meunier [Bouchon-Meunier, 1995], nous nous attardons maintenant sur le processus de raisonnement qu'il est possible d'effectuer dans ce domaine.

Il existe un nombre important d'implications floues qui portent le plus souvent le nom de leur créateur : *Kleene-Dienes* évoqué ci-dessus, *Reichenbach*, *Lukasiewicz*, etc. Ces différentes implications floues se traduisent par autant de fonctions d'appartenance distinctes, par exemple :

– Lukasiewicz : $\Phi(f_A(x), f_B(y)) = \min(1 - f_A(x) + f_B(y), 1)$.

– Mamdani : $\Phi(f_A(x), f_B(y)) = \min(f_A(x), f_B(y))$.

"Le Modus ponens*, ou détachement, est une figure du raisonnement en logique classique concernant l'implication. Elle consiste à affirmer une implication ("si* Prémisse *alors* Conclusion*") et à*

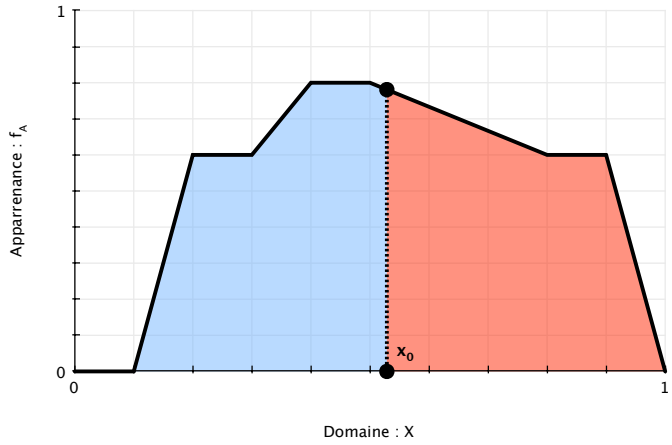

FIGURE 3.9 – Défuzzification de A : égalité des intégrales.

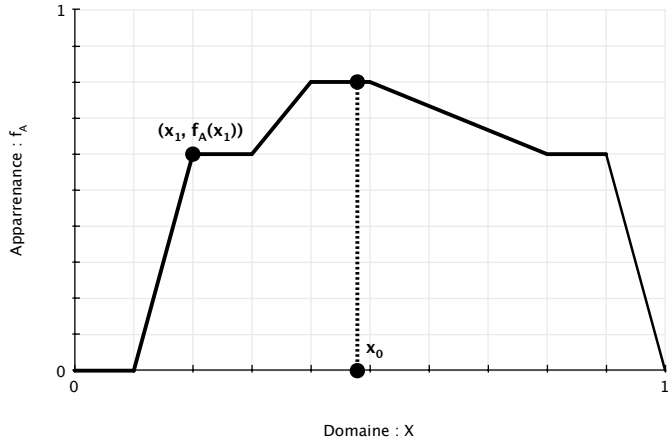

FIGURE 3.10 – Défuzzification de A : calcul du barycentre.

poser ensuite l'antécédent ("or, Prémisse *observé") pour en déduire le conséquent ("donc Conclusion"). Le terme Modus ponens (ou plus exactement* Modus ponendo ponens*) vient de ce que l'on pose A (*ponens *est le participe présent de verbe latin* ponere*, poser) afin d'en tirer la conclusion." (Wikipedia)*

Cette règle primitive du raisonnement en logique classique a été étendue aux propositions floues, elle porte alors le nom de *Modus ponens généralisé*, ou MPG [Zadeh, 1975]. Ainsi, pour deux variables linguistiques (V, X, T_V) et (W, Y, T_W), on peut poser la règle floue "V est $A \implies W$ est B" dont on peut déduire, lorsqu'elle est suivie d'une observation floue "V est A'", un conséquent "W est B'".

Dès lors que les fonctions d'appartenance f_A, f_B sont connues, ainsi que $f_{A'}$ qui représente l'observation, on va rechercher la valeur de $f_{B'}(y)$, $\forall y \in Y$. Le MPG permet en effet de combiner la règle floue avec l'observation "V est A'" pour construire la conclusion B'. L'opérateur $\top$ de MPG (une fonction de $[0,1] \times [0,1]$ dans $[0,1]$) permet de combiner $f_{\mathcal{R}}$ et $f_{A'}$: $\top$ est une T-norme liée

à $f_{\mathcal{R}}$ pour que le MPG soit compatible avec le Modus ponens de la logique classique, donc $\top$ et $\mathcal{R}$ doivent être compatibles. On a alors :

$$\forall y \in Y : f_{B'} = \sup_{x \in X} \top(f_{\mathcal{R}}(x, y), f_{A'}(x))$$

Un exemple d'opérateur de MPG, fondé sur l'implication floue de Lukasiewicz, qui pourrait alors être utilisé serait :

$$\forall u, v \in [0, 1], \top(u, v) = \max(u + v - 1, 0)$$

3.1.3 Formalismes à base de 2-tuples

L'utilisation de variables linguistiques implique un processus de calcul avec les mots pour traiter leur fusion, agrégation, comparaison, etc. Pour effectuer ces calculs, différents modèles ont été proposés, tels que le modèle sémantique [Degani et Bortolan, 1988], symbolique [Delgado et al., 1993, Truck et Akdag, 2006], ou linguistique [Herrera et Martínez, 2000].

Lors de calculs avec des nombres, le résultat obtenu est toujours exprimable dans l'intervalle de départ, étant donné que celui-ci est continu. Lors de calculs avec des mots, le résultat obtenu est une valeur artificielle ("virtual linguistic term" [Xu, 2004]) qu'il faut exprimer dans l'intervalle (nécessairement discret) de départ. Or, sauf si le résultat "tombe juste", une approximation sera nécessaire. Ainsi, certains modèles ont cherché à exprimer ce résultat sans perte de précision.

Par exemple, l'approche de Truck et Akdag envisage des 2-tuples constitués d'un symbole et d'un ensemble de symboles (ensemble discret) et gère la perte de précision en changeant itérativement la granularité de cet ensemble [Truck et Akdag, 2006]. Les auteurs expriment la précision grâce à des modificateurs symboliques.

L'approche de Wang et Hao considère des 2-tuples définis sur des intervalles continus. Ces couples sont constitués d'un terme linguistique, représenté par un SEF triangulaire, et d'une proportion symbolique [Wang et Hao, 2006]. C'est cette dernière qui endosse la perte de précision.

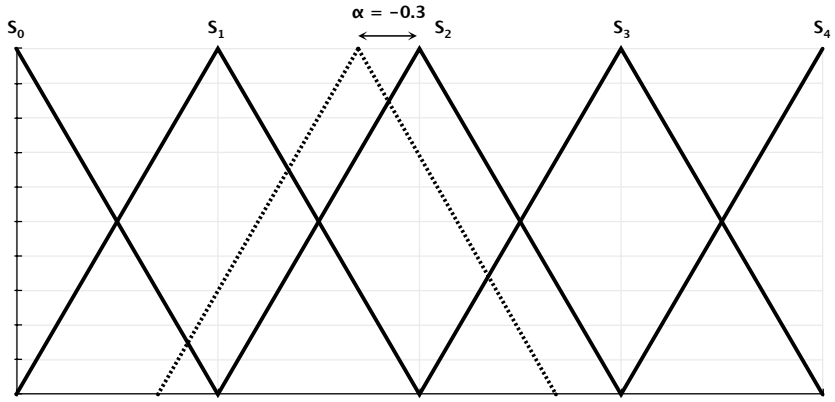

FIGURE 3.11 – Déplacement latéral d'une étiquette linguistique ⇒ 2-tuple $(s_2, -0.3)$.

L'approche de Herrera et Martínez qui a sous-tendu celle de Wang et Hao est assez similaire, sauf que les auteurs expriment l'écart entre le résultat de l'agrégation et la valeur linguistique la

plus proche dans l'ensemble de départ par une *translation symbolique* [Herrera et Martínez, 2000]. Leur modèle computationnel est fondé sur le *principe d'extension*, c'est-à-dire qu'ils étendent les agrégateurs classiques (addition, moyenne, etc.) à des agrégateurs *ad hoc*, dédiés à leurs 2-tuples. Ainsi, pour un terme linguistique donné, leur modèle fournit une paire (SEF, déplacement) $= (s_i, \alpha_i)$ avec $\alpha_i \in [-0.5, 0.5[$ tel que peut être vu sur la figure 3.11, où le 2-tuple obtenu est $(s_2, -0.3)$. De fait, grâce à la préservation du déplacement dans le tuple, ce modèle computationnel permet de conduire le processus de calcul avec des mots aisément et sans perte d'information.

Cette représentation sur intervalles continus, et la distribution uniforme des valeurs linguistiques sur l'ensemble de définition qui en découle, est la garante du modèle 2-tuple : il est ainsi toujours possible de déterminer la valeur linguistique la plus proche à partir d'un déplacement sur $[-0.5, 0.5[$. Une solution au problème posé alors par la distribution *non-uniforme* des termes sur une échelle est amenée par l'introduction de hiérarchies linguistiques qui permettent la formalisation des différentes fonctions d'appartenance hétérogènes qui leurs sont associées [Herrera et al., 2008] sur des échelles respectives où le modèle 2-tuple peut s'appliquer. Les travaux de Jiang *et al.* pour la modélisation d'environnements multi-granulaires permettent aussi de lever cet obstacle [Jiang et al., 2008].

3.2 De la modélisation compacte de préférences

C'est notamment par le développement des systèmes interactifs d'aide à la décision ainsi que par celui des systèmes autonomes dans la décision (mais qui nécessitent d'être préalablement "calibrés", nos travaux s'insérant dans cette deuxième catégorie), qu'un besoin en termes de modèles permettant de manipuler et de raisonner sur la préférence des utilisateurs s'est manifesté.

Par conséquent, des travaux en théorie de la décision se sont penchés sur la problématique de l'obtention d'un juste compromis entre deux aspects parfois opposés de la représentation compacte de ces préférences : d'une part, la nécessité de mettre au point des modèles riches en termes d'expressivité et flexibilité ; et d'autre part, un besoin plus pragmatique en termes de simplicité lors de leur élicitation, ainsi que d'efficacité lors de la résolution de problèmes d'optimisation fondés sur ces préférences. Dans notre contexte SSOA, ces contraintes sont d'autant plus présentes que les systèmes répartis fondés sur ces architectures doivent répondre d'une certaine réactivité et facilité d'utilisation. Elles nous ont alors naturellement amenés à considérer avec intérêt les modèles pré-existant de préférences linguistiques (cf. section 3.2.1) et compactes, comme les réseaux GAI (cf. section 3.2.3) ou les *CP-nets (cf. section 3.2.2).

3.2.1 Modèles linguistiques

Prendre une décision sur la base de préférences utilisateur est, le plus souvent, un problème de décision multi-critères. Or, ces dernières années, plusieurs modèles *linguistiques* de calcul ont pu être appliqués au champ de la décision sur critères multiples, et une certaine notion du concept de préférence en émerger. Herrera *et al.* établissent ainsi qu'il existe plusieurs étapes relatives à la résolution de problèmes de décision multi-critères par la méthodologie CW, lorsque l'on dispose d'informations linguistiques spécifiques [Herrera et al., 2009]. L'injection de préférences établies par des experts fait partie intégrante de ces étapes qui sont au nombre de cinq :

1. la définition du problème ;

2. l'analyse d'un problème ;
3. l'identification de solutions alternatives ;
4. l'intervention d'experts et/ou critères distincts dans la décision. C'est lors de cette étape que les préférences par lesquelles les solutions peuvent être évaluées entrent en jeu ;
5. pour finir, la sélection de la meilleure solution.

Dans le cadre d'une analyse classique d'un problème en décision floue, la dernière étape, effectuée en multi-critères ou sur la base d'un groupe d'experts, est subdivisée en deux phases distinctes [Roubens, 1997] : *l'agrégation* des valeurs de performance (d'utilité) établies sur la base des multiples critères ou experts, pour obtenir un valeur collective de performance pour chaque alternative ; *l'exploitation* de ces valeurs globales de manière à en déduire un classement des alternatives.

Il est possible de trouver un exemple de l'application des phases d'agrégation et exploitation dans les travaux de Wu et Mendel [Wu et Mendel, 2007], où le processus de décision utilisé pour la sélection des contributions à retenir pour un journal est présenté. D'autres travaux [Herrera et Herrera-Viedma, 2000] ont par ailleurs déterminé qu'il peut être nécessaire d'ajouter deux étapes préliminaires à l'application consécutive de l'agrégation et l'exploitation, de manière à obtenir concrètement un processus à trois étapes :

1. *Le choix de l'ensemble de termes linguistiques avec leurs sémantiques.* Cela consiste à établir le domaine linguistique de définition utilisé pour fournir les valeurs de performance établies entre les alternatives considérées sur critères multiples. Pour ce faire, il faut choisir la granularité de l'ensemble de termes linguistiques, ses étiquettes et sémantiques.
2. *Le choix de l'opérateur d'agrégation d'information linguistique.* Cela consiste à établir l'opérateur approprié pour l'agrégation des valeurs de performance fournies sous forme linguistique.
3. *La sélection de la meilleure alternative*, toujours découplée en une première phase d'agrégation et une seconde d'exploitation, mais qui est cette fois réalisée à partir des choix préalables en termes de domaine et d'opérateur.

Il apparaît donc comme nécessaire, pour suivre cette approche, de disposer de modèles capables de traiter la représentation et l'agrégation d'informations linguistiques : la théorie des sous-ensembles flous ainsi que les modèles symboliques et 2-tuples précédemment évoqués, peuvent remplir ce rôle. Par ailleurs, dans le cadre des 2-tuples, la problématique de l'agrégation d'informations linguistiques hétérogènes (par exemple en provenance de sources ou domaines multiples) et multi-granulaires a été étudiée, notamment par Herrera *et al.* [Herrera et al., 2005].

Par conséquent, il est effectivement possible d'exprimer, grâce à ces modèles, des préférences sous une forme linguistique (et donc qualitative) pour ensuite obtenir une décision *via* les outils du calcul avec les mots. Cependant, ces préférences ne présentent pas le même degré de flexibilité structurelle que linguistique. Plus précisément, une préférence de ce type suivra le plus souvent la forme *"Il est plus ou moins vrai que l'alternative a_k est préférée à toutes autres"* où a_k est extrait de l'ensemble des alternatives possibles $A = \{a_1, ..., a_n\}$ [Tong et Bonissone, 1980]. Il s'agit donc de préférences que l'on va qualifier *d'inconditionnelles*, dans le sens où la préférence entre les alternatives est donnée *directement* (par exemple par un expert), et ne peut dépendre d'un facteur tiers. Appliquée à notre contexte applicatif des Architectures Orientées Services, une préférence de ce type consisterait par exemple à décrire de manière fixe et définitive, avant toute prise de décision, qu'un service S_1 est préféré à un service S_2 en toute circonstance. Par conséquent, il induirait aussi la connaissance des

services disponibles avant exécution du système, ce qui n'est pas toujours applicable dans le cadre du couplage lâche entre producteurs et fournisseurs de services. On remarque par ailleurs que le caractère flou de l'expression de préférence s'applique ici uniquement au degré de confiance qu'on lui accorde.

D'autres travaux ont cherché à marier décision multi-critères et calcul avec les mots [Yager, 1981, Buckley, 1984, Shendrik et Tamm, 1986], mais ils se fondent sur un type comparable de préférences entre alternatives et un processus de décision faisant le plus souvent intervenir de manière active un ou plusieurs expert(s) du domaine. De surcroît, malgré l'abord naturel d'une représentation des préférences sous forme de phrases, les auteurs n'en proposent pas de modélisation graphique particulièrement accessible pour les utilisateurs finaux, leurs efforts étant davantage orientés vers la résolution de problématiques au demeurant plus fondamentales, telles l'hétérogénéité des préférences, leur agrégation et résolution, que vers leur élicitation. Cette absence, entièrement acceptable dans le cadre de préférences inconditionnelles, devient nettement plus aiguë dans le cadre de préférences conditionnelles de la forme *"Si condition alors preference$_1$ sinon preference$_2$"*.

3.2.2 *CP-Nets

La dénomination **CP-Nets* regroupe sous un même manteau lexical une famille de formalismes biens connus, qui sont utilisés pour l'expression de préférences utilisateur, et qui se présentent comme la solution de choix lorsque les préférences peuvent être facilement apparentées à des règles lexicographiques. Ces formalismes présentent des avantages significatifs tels qu'un abord facile pour l'utilisateur, des coûts de calcul relativement faibles, une structuration forte, et une certaine facilité d'extension, qui permet de leur adjoindre des propriétés supplémentaires en fonction de leur contexte d'application.

CP-Nets

Un CP-net ("Conditional Preference Network") est une représentation graphique et qualitative de *préférences utilisateur* [Boutilier et al., 2004]. Cette représentation est par ailleurs particulièrement compacte et relativement intuitive.

Les éléments principaux offerts par ce formalisme sont :

- les nœuds, qui représentent les variables du problème ;
- les arcs (ou cp-arcs), qui dénotent les préférences *entre* les variables en fonction des valeurs prises par ces dernières ;
- et, pour finir, des tables de préférences conditionnelles ("Conditional Preference Tables", ou CPT) directement rattachées aux nœuds et qui expriment les préférences *sur* les valeurs prises par les variables. Ce faisant, elles définissent par extension la relation binaire $>$ qui existe entre ces valeurs.

Définition 10 (CP-net)**.** *Un CP-net défini sur un ensemble de variables $V = X_1, ..., X_n$ est un graphe orienté défini sur $X_1, ..., X_n$ dont les nœuds sont annotés par des tables de préférences condi-*

tionnelles $CPT(X_i)$ pour chaque $X_i \in V$. Chaque $CPT(X_i)$ associe un ordre total $>_u^i$ *à chaque affectation u des parents de X_i telle que $Pa(X_i) = U$.*

Les CP-nets permettent la modélisation en termes de préférences de déclarations telles que "Je préfère la valeur V_1 à V_2 pour la propriété X, si une autre propriété Y vaut V_Y et Z vaut V_Z". En fait, la représentation graphique permet précisément d'exprimer la dépendance entre les CPT interconnectées par le biais de leurs variables de rattachement. Ainsi, les préférences peuvent être définies conditionnellement aux valeurs prises par leurs nœuds parents dans le graphe, mais indifféremment aux valeurs des autres nœuds.

Cette distinction constitue l'essence de la propriété dite de *ceteris paribus* ou, pour reprendre la locution latine originelle et complète, *Ceteris paribus sic stantibus*. Elle est fondamentale aux CP-nets et la locution peut se traduire, de manière générale, par "toutes choses étant égales par ailleurs". Elle est aussi utilisée en philosophie analytique, en philosophie du langage ou encore en sciences économiques, quand, dans un modèle théorique, l'influence de la variation d'une quantité (la variable explicative) sur une autre (la variable expliquée) est examinée à l'exclusion de tout autre facteur.

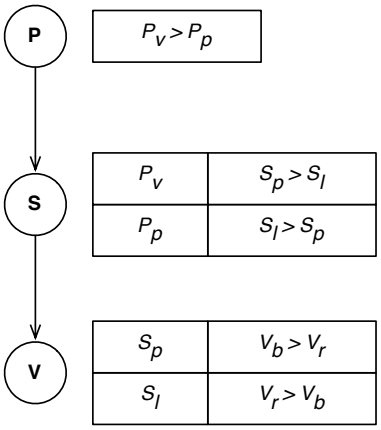

FIGURE 3.12 – Exemple de préférence sous forme de CP-net.

La figure 3.12 illustre la représentation graphique des CP-Nets à partir de l'exemple suivant : on cherche à modéliser la préférence d'une personne sur le déroulement d'un repas. Le réseau est constitué de trois variables représentant les choix possibles sur le plat principal (nœud P), la soupe (nœud S) et le vin (nœud V). Cette personne modélise alors une préférence stricte sur le choix d'un plat principal à base de viande P_v plutôt qu'à base de poisson P_p. De plus, elle préfère ne pas avoir deux plats de poisson au cours d'un même repas, de ce fait, la préférence sur la soupe est directement conditionnée par le plat principal : elle préfère alors commencer le repas avec une soupe de poisson S_p si le plat principal est à base de viande, et inversement. De la même façon, la préférence sur le choix du vin est conditionnée par la soupe qui va être servie : cette personne préfère qu'un vin rouge V_r soit servi s'il s'agissait d'une soupe de légumes S_l, un vin blanc V_b dans le cas contraire.

Il existe par ailleurs une notion de préférence relative entre les préférences elles-mêmes : une CPT associée à un nœud "parent" aura une priorité plus élevée que les CPT de ses descendants.

Cette notion de préférence relative *implicite* peut alors être prise en compte lors de la comparaison globale des affectations complètes d'un CP-net donné.

La plupart des calculs d'inférence et raisonnements logiques pouvant être effectués sur un CP-net sont praticables du point de vue de la complexité algorithmique, mais ce CP-net est alors soumis à un certain nombre de restrictions. Les principales d'entre elles (et dont il faudrait tenir compte dans notre contexte) étant l'usage généralisé de graphes acycliques, celui au contraire très limité de la relation d'indifférence entre les variables, et donc, par là même, la définition systématique de pré-ordres totaux dans les CPTs de chaque nœud parent distinct [Boutilier et al., 2004].

UCP-Nets

Les "Utility CP-nets", ou UCP-nets, différents des CP-nets dans la mesure où ils remplacent la relation binaire définie par extension entre les valeurs des nœuds, par des facteurs d'utilité sous forme numérique [Boutilier et al., 2001]. Ce faisant, les valeurs des nœuds conservent leur forme qualitative : seules les préférences, concrétisées ici par les facteurs d'utilité, sont quantifiées.

Ce changement est motivé par le fait que la précision d'une fonction d'utilité, par opposition à un ordonnancement préférentiel, est souvent nécessaire dans les contextes de prise de décision où l'incertitude est un facteur déterminant. Il est aussi motivé par le fait qu'un CP-net ne permet pas la comparaison ou l'ordonnancement de toutes ses alternatives. Cette limitation est, elle aussi, résolue par la quantification des préférences [Boubekeur et Tamine-Lechani, 2006].

Un facteur d'utilité est un nombre réel, associé à une affectation d'un nœud X du réseau, pour une affectation spécifique de ses parents. Ces facteurs d'utilité correspondent en fait aux degrés de préférence attribués, par les utilisateurs, pour les différentes affectations. En effet, grâce aux UCP-nets, un modélisateur de préférences va pouvoir se focaliser sur la définition de ces facteurs d'utilité pour exprimer sa préférence *locale* à chaque CPT, tout en se reposant sur la sémantique du formalisme pour calculer l'utilité *globale* de chaque affectation complète, de manière à pouvoir les comparer aisément.

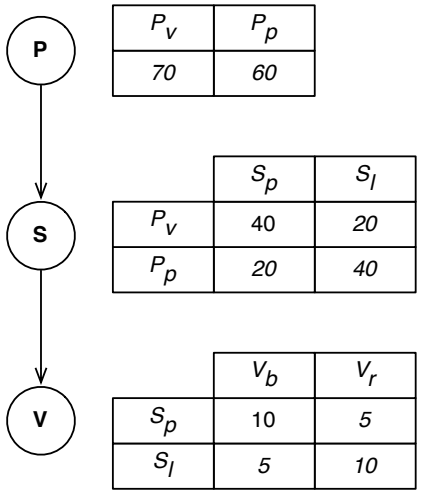

FIGURE 3.13 – Exemple de préférence sous forme d'UCP-net.

Ces concepts sont illustrés par la figure 3.13 où la préférence précédemment exprimée sous forme de CP-net à l'aide de la relation binaire $>$ est transposée sous forme de divers facteurs d'utilité qui respectent les préférences locales à chaque table, tout en rendant explicite la notion de préférence relative entre les nœuds par le choix d'un ordre de grandeur différent pour les valeurs de chaque table, en fonction de la position de son nœud respectif dans le réseau de préférences. Cet ordre de grandeur sur les valeurs doit respecter un certains nombres de critères pour que des contradictions n'apparaissent pas entre ce formalisme et le CP-nets [Boutilier et al., 2001].

TCP-Nets

Une autre extension des CP-nets, nommée "Tradeoffs-enhanced CP-nets", ou TCP-nets, permet d'exprimer des préférences de la forme "Une meilleure affectation de X est plus importante qu'une meilleure affectation de Y" [Brafman et Domshlak, 2002]. Il s'agit de *déclarations d'importance relative*.

Les TCP-nets effectuent aussi une généralisation de cette classe de préférences de manière à accepter des *déclarations d'importance relative conditionnelles*. Avec ces dernières, il devient possible d'exprimer des préférences de la forme "Une meilleure affectation de X est plus importante qu'une meilleure affectation de Y, si $Z = z$".

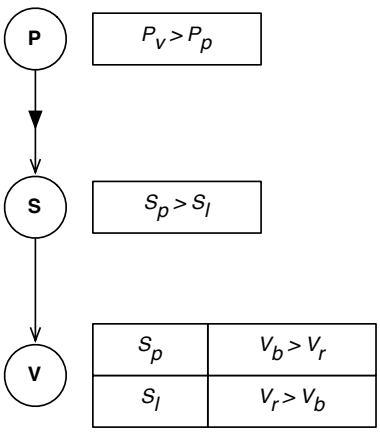

FIGURE 3.14 – Exemple de préférence sous forme de TCP-net.

Ce formalisme introduit par ailleurs une nouvelle forme de tables de préférences à l'avenant, les "Conditional Importance Tables" ou CIT, ainsi que deux nouveaux types d'arcs entre les nœuds d'un réseau de préférences : les *i-arc* et *ci-arcs*. Ils permettent, respectivement, la modélisation graphique des déclarations d'importance relative *basiques* et *conditionnelles*. L'introduction de la notion d'i-arc dans notre exemple sur la modélisation de la préférence d'un repas (cf. figure 3.14) pourrait alors servir à indiquer que l'on accorde simplement plus d'importance au respect de la préférence sur le plat principal qu'à celle portant sur la soupe, sans pour autant conditionner la préférence sur la soupe sur le type de plat principal servi. Ce deuxième point marque une différence notable par rapport à l'arc classique du formalisme CP-net de base, il est en lien direct avec la notion de "tradeoff" (*i.e.* compromis) propre aux TCP-nets.

Ainsi, les TCP-nets donnent aux utilisateurs les moyens leur permettant d'exprimer les compromis que ces derniers sont prêts à concéder entre divers critères de préférence, en fonction de l'affectation courante des variables (*i.e.* les nœuds). La notion de déclaration d'importance relative conditionnelle complète élégamment celle d'indépendance *ceteris paribus* conditionnelle, de manière à fournir un canevas conceptuel à même de permettre la modélisation et le raisonnement sur les préférences utilisateur.

3.2.3 Réseaux GAI

Les *réseaux GAI* (de l'anglais "Generalized Additive Independence", ou *indépendance additive généralisée*) [Gonzales et Perny, 2005], on parle aussi de *GAI-nets*, font partie d'une famille de langages de représentation des préférences qui, contrairement à ceux reposant sur la logique propositionnelle pour représenter des relations de manière compacte (comme les CP-nets), utilise, dans un but similaire, une propriété *d'indépendance additive entre variables* [Bouveret et al., 2005].

De manière plus générale, cette famille comporte comme autres modèles de représentation les langages de lots k-additifs [Chevaleyre et al., 2004, Grabisch, 1997] que nous n'aborderons pas dans ce chapitre car la notion d'indépendance k-additive, dans sa forme de base, ne s'applique qu'aux langages impliquant des variables binaires, là où nous avons vocation à traiter des domaines de Qualité de Service le plus souvent continus. Cependant, cette notion a une correspondance pour des langages sur des domaines combinatoires impliquant des variables non binaires : elle prend forme sous la notion d'indépendance additive généralisée, base des GAI-nets, qui a été appliquée à la représentation des utilités. L'idée est alors de transposer la notion d'indépendance probabiliste à la base du formalisme des réseaux bayésiens [Pearl et Shafer, 1988] à la représentation des fonctions d'utilité.

De fait, les *décompositions GAI* ont été introduites dans le but d'augmenter la puissance descriptive des *utilités additives*, tout comme s'y étaient déjà attelés les travaux sur l'*utilité multilinéaire* [Keeney et Raiffa, 1993].

Soit $\gtrsim$ la relation de préférence d'un utilisateur (un préordre large total) sur un ensemble χ. $x \gtrsim y$ signifie alors que x est au moins aussi bon que y. Sous certaines hypothèses [Debreu, 1964], cette relation de préférence peut être représentée par une utilité, c'est-à-dire une fonction $u : \chi \mapsto \mathbb{R}$ telle que $x \gtrsim y \Leftrightarrow u(x) \geq u(y)$ pour tout $x, y \in \chi$. Il est alors possible d'introduire la définition suivante :

Définition 11 (Décomposition GAI)**.** *Soit $\chi = \prod_{i=1}^{n} X_i$. Soit $Z_1, ..., Z_k$ des sous-ensembles de $N = 1, ..., n$ tels que $N = \bigcup_{i=1}^{k} Z_i$. Pour chaque i, soit $X_{Z_i} = \prod_{j \in Z_i} X_j$. L'utilité $u(\cdot)$ qui représente $\gtrsim$ est GAI-décomposable par rapport aux X_{Z_i} ssi il existe des fonctions $u_i : X_{Z_i} \mapsto \mathbb{R}$ telles que :*

$$\forall x = (x_1, ..., x_n) \in \chi, u(x) = \sum_{i=1}^{k} u_i(x_{Z_i})$$

où x_{Z_i} représente le n-uplet formé par les x_j, $j \in Z_i$.

Différents modèles quantitatifs (fondés sur des utilités) de représentation de préférences multi-attributs, ont précédé l'établissement de ce principe de décomposition GAI. Ces modèles se ba-

saient alors sur un type spécial d'indépendance entre les attributs nommé *indépendance préférentielle mutuelle*, qui assure que les préférences sont représentables par une utilité additive. Bien qu'une telle décomposabilité rend le processus d'élicitation de préférence simple et rapide, de récents travaux sur les GAI-nets ont avancé que cette indépendance préférentielle n'est pas nécessairement vérifiée dans la pratique, car *"elle élimine toute possibilité d'interaction entre les attributs"* [Queiroz et al., 2007]. En conséquence de quoi, ces travaux indiquent qu'une décomposition GAI permet une interaction plus générale entre les attributs tout en préservant une certaine décomposabilité du modèle [Bacchus et Grove, 1995].

Les décompositions GAI peuvent être subséquemment représentées sous forme de graphes. Ce sont ces graphes que l'on appelle réseaux GAI [Gonzales et Perny, 2004]. Ces modèles graphiques sont similaires aux graphes de jonction utilisés pour les réseaux bayésiens. C'est dans l'optique de tirer parti des caractéristiques spécifiques aux décompositions GAI que Gonzales et Perny ainsi que Braziunas et Boutilier ont alors proposé une procédure générale (en l'occurrence une séquence de questions) pour éliciter des utilités GAI dans le cadre de la décision dans le risque [Gonzales et Perny, 2004, Braziunas et Boutilier, 2005]. Cette procédure est dirigée par la structure graphique des réseaux GAI.

Définition 12 (Réseau GAI)**.** *Soit $\chi = \prod_{i=1}^{n} X_i$. Soit $Z_1, ..., Z_k$ des sous ensembles de $N = 1, ..., n$ tels que $N = \bigcup_{i=1}^{k} Z_i$. Supposons que $\gtrsim$ est représentable par une utilité GAI $u(x) = \sum_{i=1}^{k} u_i(x_{Z_i})$ pour tout $x \in \chi$. Alors un réseau GAI qui représente $u(\cdot)$ est un graphe non-orienté $G = (V, E)$ qui satisfait les propriétés suivantes :*

1. *$V = X_{Z_1}, ..., X_{Z_k}$;*
2. *pour toute arête $(X_{Z_i}, X_{Z_j}) \in E, Z_i \cap Z_j \neq \emptyset$. Pour tout X_{Z_i}, X_{Z_j}, tel que $Z_i \cap Z_j = T_{ij} \neq \emptyset$, il existe un chemin dans le graphe G qui connecte X_{Z_i} et X_{Z_j} tel que tous ses nœuds contiennent tous les indices de T_{ij} (propriété d'intersection courante). Les nœuds de V sont appelés cliques. Chaque arête $(X_{Z_i}, X_{Z_j}) \in E$ est étiquetée par $X_{T_{ij}} = X_{Z_i \cap Z_j}$ et est appelée un séparateur.*

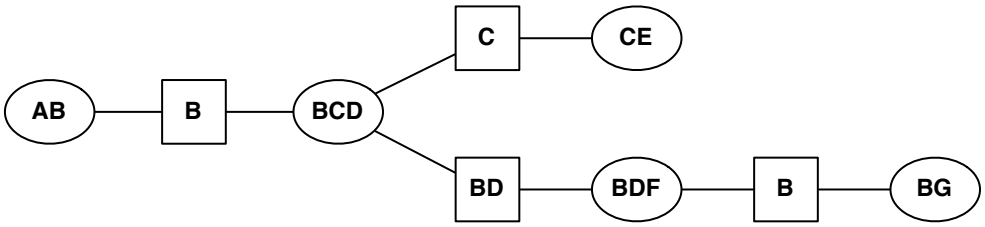

FIGURE 3.15 – Exemple de réseau GAI sous forme graphique.

D'après la précédente définition, les cliques d'un réseau GAI (correspondant à une décomposition GAI) doivent être les ensembles de variables des sous-utilités [Queiroz et al., 2007]. Les cliques étant dessinées comme des ellipses et les séparateurs comme des rectangles, on illustre sur la figure 3.15 l'exemple suivant : si $u(a, b, c, d, e, f, g) = u_1(a, b) + u_2(c, e) + u_3(b, c, d) + u_4(b, d, f) + u_5(b, g)$ alors les cliques sont *AB*, *CE*, *BCD*, *BDF*, *BG*. De plus, par la seconde propriété de la précédente définition, l'ensemble des arêtes d'un réseau GAI peut être determiné par des algorithmes qui préservent la propriété d'intersection courante [Cowell et al., 1999].

3.2.4 Synthèse

Par rapport aux réseaux GAI, la simplicité voulue des *CP-nets (qu'elle s'exprime en termes d'élicitation ou de raisonnement) est particulièrement pertinente sur le Web en général, et tout particulièrement dans le contexte des SSOA où la solution préférée doit être trouvée parmi un ensemble combinatoire de possibilités. Les *CP-nets peuvent être aisément intégrés dans des systèmes où les préférences des utilisateurs doivent être capturées en utilisant quelques questions seulement, afin de trouver rapidement parmi les éléments disponibles (des services dans le contexte SSOA) ceux qui sont les "meilleurs" selon ces préférences.

C'est dans le cadre d'autres applications où on peut consacrer plus de temps à l'étape d'élicitation, qu'il s'agisse par exemple des problèmes de configuration, de répartition équitable de ressources ou d'enchères combinatoires, et ce afin d'obtenir une description plus fine des préférences que les décompositions GAI par utilités cardinales additives peuvent s'avérer plus adaptées que les modèles purement ordinaux grâce à leur plus grande puissance descriptive supposée [Queiroz et al., 2007], et se soustraient au problème bien connu du théorème d'impossibilité d'Arrow dans le cadre de la décision collective [Pini et al., 2005].

Il ne faut cependant pas perdre de vue que l'utilisation d'utilités cardinales n'est pas l'apanage exclusif des réseaux GAI. En effet, l'intérêt de cette dernière dans l'élicitation de préférences ainsi que sa mise en œuvre ont été aussi étudiés dans le cadre des modèles compacts de préférences *CP-nets, sous la forme des UCP-nets [Boutilier et al., 2001].

Les réseaux GAI présentent par ailleurs plusieurs approches pour l'intégration de préférences émises par divers utilisateurs dans le cadre d'une prise de décision collective, *via* la recherche d'une solution de compromis [Queiroz et al., 2007]. Il s'agit cependant d'une problématique peu étudiée à ce jour dans le cadre des *CP-nets : bien qu'abordée par Rossi *et al. via* le modèle *mCP-net* [Rossi et al., 2004], elle est peu applicable aux scénarios usuels d'utilisation de nos travaux dans le contexte SSOA, tel qu'il sera présenté par un cas d'utilisation précis au chapitre 4.

Pour finir, on remarque en outre que les approches compactes de représentation graphique de préférences (*CP-nets ou GAI-nets), n'intègrent pas en leur sein une approche linguistique telle que nous l'avons définie ci-dessus et illustrée par les travaux actuels en termes de modèles linguistiques. Cette dernière permet pourtant, dans notre contexte, la prise en compte des domaines de valeurs continus d'une manière élégante. Les deux démarches évoluent donc, jusqu'à présent, dans des sphères conceptuelles distinctes, bien qu'elles partagent des objectifs applicatifs communs lorsqu'il s'agit de combler le fossé entre la connaissance métier informelle des utilisateurs d'un système d'une part, et une représentation "formelle" de cette connaissance qui soit interprétable par une machine d'autre part. Nous nous appliquerons alors à rapprocher ces deux approches dans le cadre de notre contribution pour la composition utile de services présentée au chapitre 6.

3.3 De la composition dynamique de services sous contraintes non-fonctionnelles

Par rapport à la définition très généraliste de la composition de services que nous avons fixée dans le contexte technologique (cf. section 2.1.1), la composition *dynamique* de service apporte une indication supplémentaire quant à la manière et au moment de l'effectuer. Bien qu'il n'existe pas

de définition communément admise de cette notion, les différents travaux que nous abordons dans cette section partagent le plus souvent un ensemble de caractéristiques communes qui permettent d'en déterminer les contours :

- Contrairement à une composition "classique" de services où les liaisons entre processus et services sont indiquées précisément par un opérateur, l'approche dynamique de composition relègue cette *décision de liaison* au canevas de support des SOA *via* des algorithmes capables de prendre en compte divers facteurs.
- Les contraintes non-fonctionnelles des processus ainsi que les propriétés non-fonctionnelles des services, indiquées via leurs contrats de Qualité de Service respectifs, négociés ou non (cf. section 2.3.3), constituent la majeure partie, si ce n'est la totalité, de ces facteurs.
- La *recomposition* des services à l'exécution leur permet d'effectuer les ajustements induits par les variations de QoS des services, suite à leur composition initiale et aux premières décisions de liaison effectuées en amont de l'exécution de la partie métier des processus (leur "charge utile").

3.3.1 Gestion de la dynamicité par recomposition

Dans les travaux évoqués ci-dessous, la composition de services est considérée avant tout comme une première étape "hors-ligne" de choix d'une ou plusieurs affectations globales de services aux processus métiers.

Au cours de la composition, ces travaux vont envisager, pour chaque site d'appel, l'ensemble des services disponibles. Un service est alors sélectionné pour l'affectation globale si sa QoS propre, ajoutée à celle des précédents choix, ne viole pas les contraintes imposées par le processus.

Pour le vérifier, ils font appel à des fonctions d'agrégation de QoS qui permettent d'obtenir la valeur *globale* pour un processus d'une propriété de QoS, à partir des multiples QoS *locales* des services. Schématiquement, pour un processus métier P linéaire à une seule branche (aucune alternative ni boucle), afin d'obtenir sa valeur globale A_P de QoS sur une dimension A, il faut procéder à l'agrégation des valeurs de A locales à chaque service S retenu pour l'affectation globale :

$$\forall S \in (S_1, ..., S_n), value(A_P) = \sum_{i=1}^{n} value(A_{S_i})$$

Dans cette formule, $\sum$ représente la formule mathématique utilisée pour l'agrégation des valeurs de types A.

Par exemple, pour un processus *process* dont une contrainte non-fonctionnelle globale sur le temps maximum t à respecter a été fixée :

$$t_{process} \leq 40ms$$

Plusieurs affectations globales peuvent être déterminées (cf. figure 3.16). L'exécution du processus sur la base d'une de ces affectations est alors sensée respecter ses contrainte de QoS.

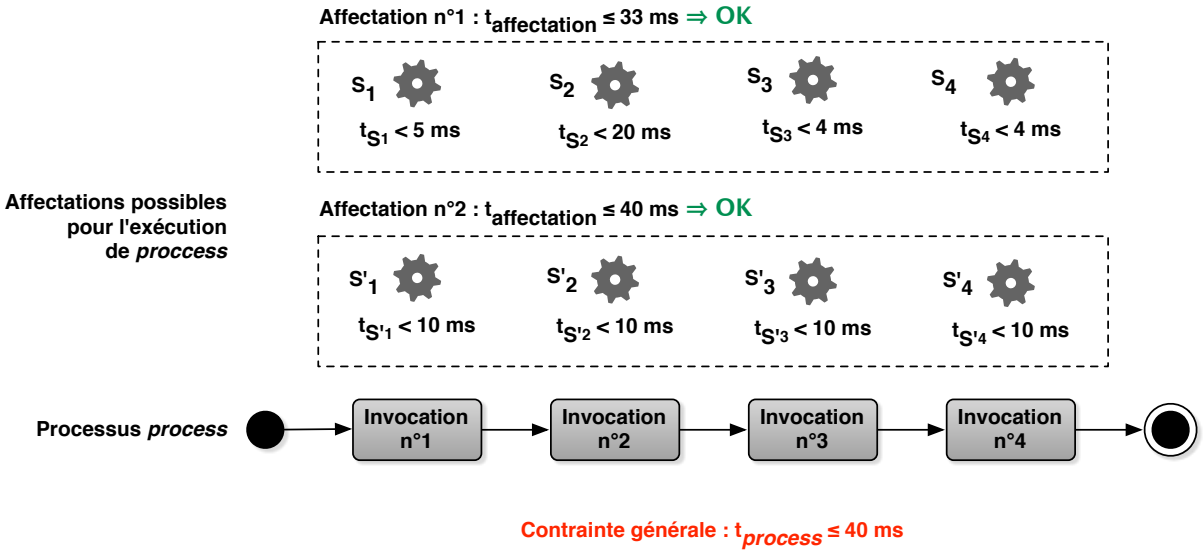

FIGURE 3.16 – Composition dynamique : calcul des affectations.

Les travaux considérés ci-dessous, proposent leurs propres définitions de $\sum$ pour chaque type de QoS qu'ils traitent. Les plus avancés ne se limitent plus à une approche linéaire, mais intègrent les instructions de gestion du flot de contrôle (utilisées habituellement dans un processus métier) au sein du calcul d'agrégation : par exemple, via l'utilisation de moyennes des valeurs de QoS pour traiter les branches conditionnelles.

Le caractère particulièrement coûteux d'une évaluation exhaustive et *a priori* de toutes les combinaisons possibles d'affectations renforce le caractère statique de ces approches de composition. Ces travaux proposent alors de les compléter par un processus de *recomposition*, déclenché lors de l'exécution des processus métiers : il s'agit de procéder à une nouvelle affectation de services aux différents sites d'appels du processus, de manière à respecter à nouveau les contraintes non-fonctionnelles. Son coût varie en fonction de l'étendue de la recomposition : pour une recomposition de tout le processus, il est total [Ben Mokhtar et al., 2007, Canfora et al., 2006] ; pour une "reconfiguration" limitée, il peut être minimisé [Colombo et al., 2006, Halima et al., 2008].

Canevas COCOA

Le canevas COCOA ("COnversation-based service COmposition in pervAsive computing environments") [Ben Mokhtar et al., 2005, Ben Mokhtar et al., 2007] applique les principes de composition et recomposition de services au contexte de l'informatique omniprésente.

Il prévoit un ensemble d'agrégateurs capables de traiter la *disponibilité*, *latence* et le *coût* des services en fonction de certaines structures précises d'un processus métier (boucles, tests conditionnels, etc.). Le canevas va faire appel à ces formules lors de la composition puis de la recomposition. Cette recomposition est effectuée à partir des services disponibles dans son environnement, lorsqu'une violation de QoS globale est détectée à l'exécution.

Ces travaux adjoignent au processus de sélection de services un raisonnement logique fondée sur une ontologie de type OWL-S. Les concepts ontologiques utilisés par les consommateurs de services, corrélées à ceux des fournisseurs de services, vont alors influencer la composition de services au même titre que les valeurs de QoS. Les ontologies fixent en effet un cadre pour la définition de nouvelles caractéristiques non-fonctionnelles par extension du modèle de base proposé par COCOA. Dans ces travaux, les ontologies OWL-S ont même vocation à décrire l'ensemble d'une offre ou requête de service, et pas seulement les concepts métiers utilisés pour annoter des offres de type SAWSDL (cf. section 2.1.3). Cependant, on s'aperçoit que la norme SAWSDL s'est vite imposée comme spécification dominante dans les plus récents travaux sur les SSOA, ce qui va probablement induire une mise-à-jour de ces travaux sur le long terme.

Canfora *et al.*

Les récents travaux de Canfora *et al.* mettent en avant une approche globale pour traiter la problématique de la composition dynamique des services [Canfora et al., 2005b, Canfora et al., 2006]. Elle se décline, d'un point de vue architectural, en un évaluateur de QoS ("QoS evaluator") et un composant de liaison ("binder").

La composition y est réalisée sur la base de caractéristiques à la frontière du fonctionnel et du non-fonctionnel, le plus souvent liées aux domaines métiers. Pour définir ces caractéristiques, les auteurs introduisent un nouveau langage pour la description des propriétés de QoS ainsi qu'un second langage dédié, pour définir les formules mathématiques nécessaires à l'agrégation des données, inspiré des précédents travaux de Cardoso [Cardoso, 2002]. L'*évaluateur de QoS* mis au point est alors chargé d'interpréter les définitions formulées dans ces deux langages.

La problématique de la complexité *NP-difficile* de l'évaluation exhaustive des combinaisons possibles d'affectations de services est traitée, quant à elle, par l'utilisation d'algorithmes génétiques [Canfora et al., 2005a] là où d'autres auteurs ont pu déployer des techniques de programmation en nombres entiers ("ìnteger programming") [Zeng et al., 2004].

Le principe de recomposition est ici décliné sous le vocable de "re-binding" des services et de "replanning" des processus. En partant de la constatation de la volatilité des valeurs effectives de QoS et de leur évolution par rapport à celles contractées lors de la phase de composition, les auteurs ont mis au point un composant capable d'effectuer, *dans certains cas précis*, une *recomposition localisée* des services au cours de l'exécution des processus métiers. L'idée est donc de se soustraire au coût particulièrement important du calcul complet d'une nouvelle composition en limitant la recomposition à la branche (ou tranche, "slice") spécifique du processus en cours d'exécution. La recomposition va alors être déclenchée :

- *a posteriori*, après la constatation d'une violation par un service précédemment composé de ses engagements en termes de QoS ;
- dans certains cas, de manière préventive avant que la violation ait lieu, si elle est considérée comme hautement probable.

SCENE

Le canevas SCENE ("Service Composition ExecutioN Environment") consiste en une architecture logicielle et un langage défini comme une extension de BPEL [Colombo et al., 2006]. Le principe de fonctionnement du canevas consiste à permettre dans un premier temps la description des politiques/méthodes de *reconfiguration* qui seront déclenchées suite à la violation d'une ou plusieurs "règles". La reconfiguration est en fait une terminologie utilisée dans ces travaux pour désigner une *recomposition partielle* d'un processus, donc *a priori* moins coûteuse. Il est par ailleurs possible de comparer le degré d'expressivité de ces règles à celui des contraintes non-fonctionnelles exprimables dans le langage QML (cf. section 2.3.3).

A l'exécution, c'est grâce à l'architecture logicielle spécifiquement mise au point que les violations sont détectées et les politiques de recomposition interprétées et déclenchées. Il s'agit donc ici d'orienter la reconfiguration via ces politiques, et non la composition initiale des services effectuée en amont.

Halima *et al.*

C'est une seconde approche pour la reconfiguration qui est cette fois proposée par Halima *et al.* [Halima et al., 2008]. Elle est particulièrement dynamique dans la mesure ou les auteurs préconisent d'observer les messages échangés au sein d'une application répartie fondée sur des services Web pour procéder à des opérations de reconfigurations ponctuelles, limitées dans dans le temps et dans l'espace (ce n'est que dans un cas extrême que la totalité d'un processus aurait à être recomposé).

Tout comme dans les précédents travaux, la reconfiguration passe par la définition préalable de politiques nécessaires à sa (bonne) conduite, avec la particularité toutefois que ces politiques disposent ici du moyen de provoquer jusqu'au déploiement de nouveaux services sur le réseau.

Par cette dynamicité, ils introduisent la notion d'"auto-réparation" des processus, qui reste cependant en grande partie à la charge du canevas de support et non du processus lui-même, comme pourrait le laisser entendre cette dénomination. On remarque toutefois que ce type de travaux se situent à la frontière de la notion de recomposition telle que précédemment définie, et une démarche plus *active* comme préconisée dans l'introduction de ce manuscrit. Ils restent cependant prisonniers d'une logique de traitement des pannes et non d'adaptation agile aux fluctuations non-fonctionnelles des services qui les composent.

WSQoSX

Les travaux menés par Berbner *et al.* sur l'architecture WSQoSX ("Web Service Quality of Service Architectural Extension"), reposent sur le principe d'une sélection de services réellement dynamique, effectuée lors de l'orchestration d'un processus métier, via un composant proxy qui s'intercale entre le moteur d'orchestration et les services disponibles [Berbner et al., 2007].

Cette sélection est menée sur la base des informations de QoS définies dans les contrats rattachés aux services disponibles, et non sur la QoS courante des services au moment de leur sélection. Si le service ne respecte pas son contrat, donc en cas d'erreur, une nouvelle sélection de service, locale au site d'appel du processus, est effectuée. Ce mécanisme est complété par une possibilité de

"replanning" (donc de recomposition) du processus, si ce premier n'est plus suffisant pour absorber les variations de QoS des services qui avaient été considérés pour la composition.

3.3.2 Composition par génération de processus alternatifs équivalents

Chiu *et al.*

La composition dynamique sur critères non-fonctionnels proposée par Chiu *et al.* est limitée aux deux dimensions *"accuracy"* et *"time"*, qui sont intégrées dans le processus décisionnel au moment d'une composition effectuée avant exécution [Chiu et al., 2009].

Cette composition de services est donc fondée sur un ensemble très restreint de dimensions pré-établies de QoS, dont l'agrégation des valeurs est effectuée grâce à un jeu de formules mathématiques comparables à celles mises en place dans COCOA.

Ces travaux sont axés sur la génération de "workflows" : ils cherchent ainsi à établir à l'avance plusieurs *processus alternatifs* capables de répondre à un même besoin fonctionnel décrit de manière abstraite, de manière à pouvoir en changer à l'exécution si les contraintes non-fonctionnelles l'exigent.

Chafle *et al*

C'est une approche similaire que l'on retrouve chez Chafle *et al.* : ces travaux cherchent à limiter l'impact de la recomposition de services par le précalcul hors-ligne de plusieurs processus métiers similaires fonctionnellement et qu'il est possible de classer en fonction de leur pertinence par rapport aux contraintes non-fonctionnelles initiales [Chafle et al., 2006].

En cas de panne, l'algorithme mis au point par les auteurs passe au processus disponible suivant, et, s'il n'en reste plus, fait appel à une coûteuse recomposition globale.

Canevas CoRE

Tout comme pour COCOA, l'utilisation d'informations "sémantiques" est mise en œuvre dans un contexte SOA par le récent canevas CoRE ("Component Runtime Environment") [Fujii et Suda, 2009].

Cette mise en œuvre s'effectue alors au travers de son composant SeGSeC ("Semantic Graph based Service Composition"), mais à beaucoup plus grande échelle que dans COCOA, car il s'agit ici de créer un ou plusieurs processus métier alternatifs de toute pièce, à partir de requêtes d'utilisateurs formulées en langage naturel. Ces travaux avancent que la création des alternatives est facilitée par la sémantique métier disponible et se fonde sur un calcul de correspondance entre concepts pour y parvenir.

Ce type d'approche fondée sur le langage naturel est particulièrement tributaire de la richesse des informations fournies par les utilisateurs, de celles disponibles dans le "contexte" de l'application répartie, et du nombre de services disponibles. En effet, dans la mesure où un squelette de processus métier n'est pas établi par l'utilisateur avant la phase de composition de services, le composant SeGSeC va nécessiter un nombre important de services fonctionnellement équivalents pour atteindre son plein potentiel de création de processus métiers : chacun de ces services constitue une alternative

possible pour répondre aux besoins des utilisateurs et créer un enchaînement logique fondé sur la sémantique.

3.3.3 Composition dynamique dirigée par CP-nets

Schröpfer *et al.*

L'idée d'apparier CP-nets et propriétés non-fonctionnelles, de surcroît pour effectuer de la composition dynamique de services, a déjà fait l'objet d'une étude par Schröpfer *et al.*, les auteurs y définissent des préférences au travers de CP-nets, de manière à pouvoir sélectionner le "meilleur" service sur la base de sa QoS [Schröpfer et al., 2007]. L'utilisation de TCP-nets leur permettent notamment d'intégrer la notion de compromis qui peut s'avérer indispensable pour l'expression de préférences véritablement utiles.

Par rapport à notre problématique (cf. 1.2) et l'approche que nous préconisons, il semble cependant que deux points manquent dans leur démarche : l'intégration de la notion de QoS courante des services lors du processus décisionnel (ils se fondent sur les contrats non-fonctionnels uniquement), ainsi que la définition explicite d'un procédé permettant de traiter convenablement les domaines par nature continus de QoS (tels que la latence, la bande passante, etc) dans leur représentation qualitative. Pour gérer ces valeurs, ils sont soumis aux même contraintes que celles des UCP et TCP-nets sur lesquels ils reposent : il est nécessaire d'effectuer un partitionnement des domaines strict, aux frontières artificielles, de manière à obtenir des variables manipulables dans les tables de préférences par nature discrètes des TCP-nets (par exemple : si $0 \leq$ bande passante < 10 alors on l'appelle *bande passante faible*).

Par ailleurs, comparativement aux précédents travaux sur la recomposition, ceux menés par Schröpfer *et al.* se focalisent bien plus sur l'étape de la sélection des services sur la base de préférences que sur la mise en place du canevas dans lequel s'insère cette sélection. Il apparaît donc comme nécessaire de les intégrer aux d'autres approches plus globales pour pouvoir les exploiter correctement.

TCP-Compose*

Les travaux menés par Santhanam *et al.* ont consisté à mettre au point un algorithme *TCP-Compose** pour la composition "efficace" de services Web sur la base de préférences qualitatives. Pour arriver à cette fin, ils se fondent sur les réseaux de type TCP-net. [Santhanam et al., 2008], ce qui leur permet de disposer de modèles dont les propriétés sont assez similaires à ceux de Schröpfer *et al.*.

Cependant, leur algorithme propose de s'atteler au calcul d'une composition dans son ensemble et non, contrairement à ces derniers, à celui d'une sélection individuelle de service. Cette approche ambitieuse repose alors sur le *calcul de dominance* des affectations d'un TCP-net dont la définition doit englober toutes les caractéristiques non-fonctionnelles considérés de la composition.

Cette démarche a néanmoins un impact non négligeable sur le caractère dynamique de la composition puisque celle ci devient insécable par la nature même de l'algorithme de comparaison TCP-Compose* et du calcul de dominance sur CP-nets. Elle devrait donc être probablement accompagnée d'une recomposition, même si celle-ci n'est pas encore envisagée par les auteurs.

3.3.4 Synthèse

Les différentes approches pour la composition dynamique de services sous contraintes non-fonctionnelles que nous avons abordées dans cet état de l'art présentent des fonctionnalités évoluées et sont le résultat d'une communauté scientifique toujours très active. Cependant, par rapport à notre problématique précise de mise en œuvre d'une composition agile de services, leurs limitations sont le plus souvent similaires : la composition des services est effectuée relativement tôt dans le cycle de vies de processus métiers et la mise en avant de la recomposation, comme principe d'adaptation aux évolutions du contexte à l'exécution, peu pertinente dans le cas où les caractéristiques non-fonctionnelles des services varient au sein de bornes de QoS définies comme acceptables dans les contrats de Qualité de Service : dans ce cas, les "erreurs" qui servent de déclencheurs à la recomposition ou la reconfiguration ne sont pas levées, ou alors seulement après accumulation. C'est ce constat qui va mener à la distinction entre composition *active* et composition *dynamique* (cf. section 5.2).

3.4 Conclusion

Nous avons effectué, dans ce chapitre, un état de l'art des travaux connexes aux contributions de cette thèse. Concernant l'*approche linguistique pour raisonner* et la *modélisation compacte de préférences*, il apparaît qu'une combinaison des deux approches au sein d'un même formalisme n'ait pas encore été, à ce jour, effectuée et appliquée à notre problématique de prise en compte des contraintes non-fonctionnelles lors de la composition de services : elle apporte pourtant la promesse d'une meilleure prise en charge des préférences des utilisateurs en fournissant les outils formels permettant notamment de franchir convenablement le fossé séparant domaines continus de QoS et termes linguistiques du langage courant, flous par nature. C'est cette démarche qui sera mise en œuvre lors de la *composition utile* de services (cf. chapitre 6).

A l'issue de l'état de l'art sur la *composition dynamique de services sous contraintes non-fonctionnelles*, apparaît une absence certaine de la prise en charge des multiples variations de la QoS *courante* des composants (services dans le contexte SSOA) d'une application répartie, quand ceux si se situent *sous* le seuil de déclenchement d'une erreur bloquante pour la bonne marche processus métier en cours d'exécution. L'intégration dynamique de ces valeurs courantes, conditionnée par les préférences des utilisateurs, et ce pendant l'orchestration des services, aurait pourtant comme avantage de permettre une meilleure adaptivité des applications à leurs milieux. Cette adaptivité va d'ailleurs s'avérer particulièrement utile au cas d'utilisation de nos travaux (cf. chapitre 4). Par ailleurs, cette technique présenterait l'opportunité de palier, si possible, l'accumulation de petites déviations locales des propriétés non-fonctionnelles des services orchestrés. Ce constat est à la base de notre contribution en terme de *composition active* des services (cf. chapitre 5).

Chapitre 4

Un cas d'utilisation en gestion de crise environnementale

Sommaire

Ce cas d'utilisation porte sur une facette particulière de la gestion de crise environnementale : la lutte anti-incendie dans un contexte civil, sur des feux de grande envergure en milieux naturels (forêt, pinède, etc.). De par ses caractéristiques intrinsèques, nous verrons qu'elle constitue un support idéal à la présentation des contributions scientifiques identifiées dans ce manuscrit, à savoir : la composition active (cf. chapitre 5), utile (cf. chapitre 6) et agile de services (cf. chapitre 7).

Qui plus est, force est de constater que des feux de ce type surviennent de plus en plus fréquemment sous nos latitudes, conséquences directes de la combinaison de multiples facteurs climatiques et humains. Pour pouvoir lutter efficacement, ils imposent le déploiement d'importants moyens et, plus que tout, une composition performante et "intelligente" des ressources terrestres et aériennes présentes sur le théâtre des opérations.

4.1 Introduction

La préservation des ressources naturelles constitue depuis quelques décennies un nouveau défi mondial. C'est en sa qualité écologique et même économique que l'on cherche alors à sauvegarder "l'or vert" : les espaces naturels comme les forêts sont des réservoirs innés de CO_2 dont on cherche dorénavant à limiter la libération massive dans l'atmosphère lors de leur combustion ; ils constituent

par ailleurs de nouveaux pôles touristiques inégalements répartis entre les pays. Ainsi, sur le "vieux continent", les feux de grande ampleur en milieu naturel, induisent la mise en place rapide et efficace de moyens divers et variés afin de constituer une réponse appropriée pour la sauvegarde du milieu ainsi que la défense des biens et des personnes : en France, on ne lutte plus contre les incendies comme au siècle passé, de nombreuses ressources sont, sauf cas exceptionnels, mises à disposition par les secours. Le réel défi est donc d'implanter des approches innovantes pour la composition de ces ressources afin d'en tirer *au mieux* parti.

Dans ce contexte, et par rapport aux approches usuelles de gestion des crises environnementales, nous avançons qu'une amélioration du traitement de la situation peut être obtenue par l'utilisation en partie automatique et dynamique des différents moyens disponibles pour l'intervention. Le but de nos travaux n'étant pas de se substituer à la chaîne de "commande et contrôle" classique qui nécessite l'intervention humaine, mais plutôt de compléter cette action par l'automatisation de certaines procédures ciblées. Il existe en effet dans ces métiers, de nombreux processus d'intervention, ou protocoles, fruits de nombreuses années de pratique sur le terrain, qu'il est raisonnable de suivre scrupuleusement. Cette "formalisation" des processus s'est d'ailleurs accentuée ces dernières années sous l'impulsion de nombreuses initiatives de standardisation européennes. A l'aune de cette automatisation croissante, il faut cependant considérer que la disponibilité effective des moyens, de par leurs affectations simultanées sur différents théâtres d'opération du territoire, n'est souvent connue que lorsque la crise survient, ce qui à pour conséquence de déjouer toutes tentatives de "provisionnement". A cette disponibilité *initiale* au lancement d'un processus, qui conditionne la mobilisation des ressources, s'ajoute celle de leur disponibilité *au cours de son exécution*. En effet, ce cas d'utilisation présente une caractéristique supplémentaire de dynamicité du fait que certain des services correspondant à des ressources physiques engagées sur le terrain "s'usent" avec le temps : la disponibilité doit alors être vue comme une caractéristique de Qualité de Service évoluant au cours du temps, et doit être intégrée comme telle dans le processus décisionnel. Par conséquent, un haut niveau d'agilité est nécessaire, au cours de la crise, pour assurer une mobilisation pleine et entière des moyens effectivement disponibles à tout instant.

On voit aussi apparaitre, dans ce contexte, un besoin de gestion des priorités d'intervention, et cela à différents niveaux. Par exemple, une hiérarchisation doit être effectuée entre une défense renforcée des biens, des personnes ou du milieu naturel : dans le cas d'un feu de forêt, éloigné des villes et des habitations, il paraitra logique de favoriser une défense accrue du milieu, au vue des risques minimes ou inexistants encourus pour les personnes ou les biens. Ce choix ne pourra être effectué qu'à partir du moment du déploiement des effectifs, en fonction des caractéristiques de la crise, et n'est pas "pré-câblé" dans les protocoles d'intervention. Il peut alors être assimilé à des *préférences non-fonctionnelles des utilisateurs* ; l'utilisateur étant, en l'occurrence, un responsable d'un centre de commande. On met ainsi en exergue un rapprochement direct entre cette notion de priorité et celle d'*utilité* des ressources mobilisées : les ressources les plus utiles à la résolution de la crise, selon la sémantique de l'utilité propre aux préférences déployées, seront traitées comme prioritaires lors de la composition des services.

On cherche ainsi tout particulièrement à illustrer, *via* ce cas d'utilisation, l'importance de :

- La composition des différents intervenants, qui doit se traduire par l'utilisation dynamique des moyens mis à disposition.
- La séparation entre deux moments principaux d'intervention. En premier lieu, la spécification *avant la crise* de processus métiers correspondants à des protocoles usuels, ainsi que celle, sous

forme de services, des ressources (ou intervenants) que l'on pourra interroger et commander à distance. Vient ensuite, *lors de la crise*, l'exécution de ces processus métiers en fonction des divers services effectivement disponibles lors de leur lancement, découverts dynamiquement puis sélectionnés lors de l'exécution des processus sur la base de leurs propriétés non-fonctionnelles courantes.

– La prise en compte de cette QoS courante des services lors de leur orchestration, sous le prisme des préférences utilisateur, de manière maximiser l'utilité (et donc l'efficacité) globale des processus.

4.1.1 Contexte d'intervention

Suite à un incendie de grande envergure en zone forestière, montagneuse de surcroît, les soldats du feu et leur matériel sont amenés à se déployer pour mener à bien des opérations de prise de contrôle de la situation. Une des clé de la résolution correcte de cette crise réside alors dans la composition active et utile des diverses ressources mises à disposition, effectuée notamment sur la base d'informations obtenues à partir de moyens de reconnaissance aériens commandés à distance.

FIGURE 4.1 – Localisation de la zone sinistrée.

Plus précisément, nous situons ce scénario, en région Provence - Alpes - Côte d'Azur (cf. figure 4.1, connue pour ces incendies forestiers saisonniers et son relief difficile qui induit le plus souvent l'utilisation de matériel spécifique, tel des drones, pour effectuer des missions de reconnaissance.

4.1.2 Phases principales en gestion de crise

A partir de la liste donnée ci-dessous des principales phases de traitement communément admises dans le milieu de la gestion de crise, on précise que ce cas d'utilisation intervient dans les première et troisième phases : c'est en effet pendant la phase de *supervision* (hors crise) que sont établis ou raffinés les processus de gestion d'incendies, et lors de la *réponse* qu'ils sont mis en œuvre.

1. ⇒ **Supervision** : veille en continue à partir de systèmes de mesure déjà en place.
2. **Alerte** : un incendie est détecté par les systèmes de supervision, une réponse doit être engagée.
3. ⇒ **Réponse** : Les différents intervenants doivent être déployés sur site. Une composition doit être mise en place, afin de répondre à la crise.
4. **Post-Crise** : une fois la crise traitée, retour à la normale en phase de supervision. Une évaluation des dommages peut être effectuée à partir des données récoltées pendant la phase de réponse, ainsi qu'une remise en état des ressources engagées, ce qui peut avoir un impact sur la réponse suivante si elle doit être engagée rapidement après celle venant de se terminer.

4.2 Des liens étroits avec les SOA Sémantiques et nos travaux

En tout premier lieu, une particularité de l'approche mise en œuvre dans ce cas d'utilisation est d'effectuer un rapprochement entre la notion de "moyen" ou "ressource" et celle de service, au sens SOA du terme. En effet, à chaque ressource considérée (par exemple drone, camion, pompier, etc.) est associée une représentation sous forme de service sur le réseau interne du système d'intervention. Les interfaces de chacun de ces services permettent de leur envoyer des ordres, et leurs offres de service publiées dans un registre à disposition de tous, exposent aussi bien leurs caractéristiques statiques fonctionnelles que non-fonctionnelles. A cette utilisation des services pour représenter les moyens, s'ajoute celle de processus métier pour spécifier les protocoles de lutte anti-incendie qui exploiteront, à l'exécution, ces services.

Les principales caractéristiques des SOA Sémantiques apportent également des éléments de réponse pour faire face aux problématiques courantes d'interopérabilité et de composition de services dans un contexte de gestion de crise. Ces éléments de réponse se fondent sur :

- La formalisation des connaissances propres à chaque corps de métier sous forme de bases de connaissances (par exemple ontologies). On obtient ainsi un modèle sémantique de chaque domaine, voire un modèle de connaissances commun à tous les acteurs.
- L'utilisation de ces connaissances de haut niveau lors de la spécification des offres de services et de leurs demandes dans les processus métiers, sous forme de méta-données.
- L'exploitation de ces méta-données à l'exécution, ce qui permet d'élargir considérablement le champs et la pertinence de la recherche de services lorsque la crise survient : la jonction entre un producteur et un consommateur de besoin est alors effectuée dynamiquement sur la base de critères sémantiques plus souples que les critères syntaxiques.

Cependant, si l'approche SSOA par elle-même amène effectivement un niveau certain d'agilité lors de la réponse à la crise, elle n'est pas sans contraintes et doit être consolidée pour être réellement

exploitable sur le terrain. Plus spécifiquement, il ne suffit pas d'exploiter les informations sémantiques une fois pour toute lors du filtrage statique des services disponibles dans un annuaire, il faut au contraire exploiter l'élasticité du lien entre producteurs et consommateurs de service pour y ajouter de la dynamicité. En réponse à ces limitations, nous proposons alors :

- l'implantation d'une architecture pour l'orchestration véritablement dynamique des services, car ces derniers peuvent apparaître et disparaître à tout moment lors de l'exécution (nouveaux arrivants, destructions de matériel, etc.) ;
- la prise en compte des informations courantes de Qualité de Service lors de l'exécution des processus métiers, et la mise au point d'outils pour discriminer efficacement entre les services disponibles sur la base de ces informations. Cette approche permet alors, *via* la satisfaction des préférences spécifiées par les utilisateurs, une optimisation des délais d'intervention et de la répartition des ressources.

4.3 Déroulement du cas d'utilisation en lutte anti-incendie

On segmente ce cas d'utilisation en plusieurs sous-parties, ou *scenarii*. Les premiers correspondent aux actions qui ont dû être effectuées en amont de la phase de réponse : il s'agit de la définition statique des offres de services avant l'exécution (cf. 4.3.1), et de celle d'un processus métier qui sera exploité ultérieurement (cf. 4.3.2). Les *scenarii* 4.3.3 et 4.3.4 correspondent quant à eux respectivement à l'élicitation des préférences de l'utilisateur en fonction des caractéristiques immédiates de l'incendie puis, à l'orchestration dynamique des services lors de la crise.

Il est important de noter les deux premiers *scenarii* s'inscrivent dans une démarche SSOA usuelle et ne mettent pas en avant, à proprement parler, les contributions scientifiques de cette thèse. Cependant, ils sont nécessaires dans la mesure où ils mettent en place le substrat dans lequel nos travaux peuvent évoluer, à savoir : un couplage lâche entre offres et demandes de services et la possibilité d'arbitrer entre de nombreux services équivalents fonctionnellement. Par ailleurs, le but de ce chapitre étant d'articuler les concepts implantés par nos travaux autour d'un cas d'utilisation simple mais concret, nous y limiterons, autant que possible, la présence des détails technologiques et techniques afférents.

4.3.1 Scénario 1 : définition des offres services

Lors de l'adoption d'une architecture SSOA dans ce contexte de commande et de contrôle, une des premières tâches à effectuer consiste à assurer la présence effective, sous forme de services, des personnels et matériels à interroger, commander ou contrôler à distance. Dans le cadre de ce scénario et des suivants, on se limite à la distinction entre quatres types de services : les camions citerne d'intervention anti-incendies, les drones de surveillance du théâtre des opérations, et pour finir, les pompiers et gendarmes dont les équipements de nouvelle génération permettent la transmission efficace d'information et la remontée de valeurs et provenance de capteurs environnementaux et physiques.

Une fois réglées les contraintes techniques liées au déploiement de cette nouvelle technologie, l'étape la plus importante consistera alors à spécifier les caractéristiques publiques, ou offres, des

services. La description d'une offre de service, telle que visible dans un annuaire de service, comprendra trois facettes :

1. La spécification des *caractéristiques techniques* du service sous forme syntaxique. Il s'agit des protocoles, méthodes, entrées/sorties... en l'occurrence tout ce que l'on retrouve habituellement dans une offre de service. Ces caractéristiques se matérialiseront dans le cas présent sous la forme d'offres au format WSDL, ce dernier constituant un standard de fait dans les SOAs.

2. La spécification des *caractéristiques fonctionnelles* du service, sous forme d'annotations sémantiques apposées sur l'offre de service WSDL grâce à une extension récente de la spécification nommée SAWSDL.

3. La spécification des *caractéristiques non-fonctionnelles* du service. Pour cette spécification dite "statique", car effectuée avant l'exécution du système, il s'agit de poser des garanties sur des bornes minimales et maximales de QoS que le service s'engage à fournir.

Pour chacun des quatre types de service retenus dans ce cas d'utilisation, on définit individuellement dans ce scénario une offre "générique" que chaque service effectif du type concerné sera tenu de respecter. L'enregistrement dans un annuaire d'offres identiques pour tous les services d'un même type découle d'une contrainte de concision du cas d'utilisation, il ne correspond pas à une limitation technique. Cependant, il est possible d'y trouver une certaine vraisemblance dans un contexte réel de déploiement si l'on considère que tous les matériels utilisés partagent des "caractéristiques constructeurs" similaires, et donc des offres générales, et génériques, identiques. Par ailleurs, à chaque service disponible lors de l'exécution du système sera systématiquement associée une unique offre de service et, inversement, à chaque offre de service sera associé un seul et unique service.

Ainsi, pour illustrer ce scénario par une démarche toujours concise, on s'intéresse ici aux deux dernières facettes des offres de service, c'est-à-dire à la spécification de leurs caractéristiques fonctionnelles et non-fonctionnelles, où interviennent plus directement nos travaux.

Camions citerne d'intervention anti-incendies

Pour une offre de service représentant un camion citerne d'intervention, on l'annote au niveau global par le concept fonctionnel *Camion_Citerne_Anti_Incendies* extrait d'une ontologie métier disponible par ailleurs, grâce à l'attribut *sawsdl :modelReference* (cf. figure 4.2). On l'annote aussi au niveau des ses méthodes par le même procédé : l'offre expose ainsi une méthode permettant de communiquer une destination à atteindre et une seconde pour indiquer un emplacement à arroser, toutes deux annotées grâce à d'autres concepts ontologiques disponibles (*aller_a* et *arroser*) et connues des producteurs et consommateurs de services. Il est à noter que si ces annotations se limitent dans cet exemple à la granularité de la méthode, il est tout a fait envisageable, voire souhaitable, d'en annoter aussi les différents paramètres et types de données. Le tableau suivant donne l'équivalence entre la représentation sous forme de pseudo-code de ces méthodes et leur concrétisation SAWSDL.

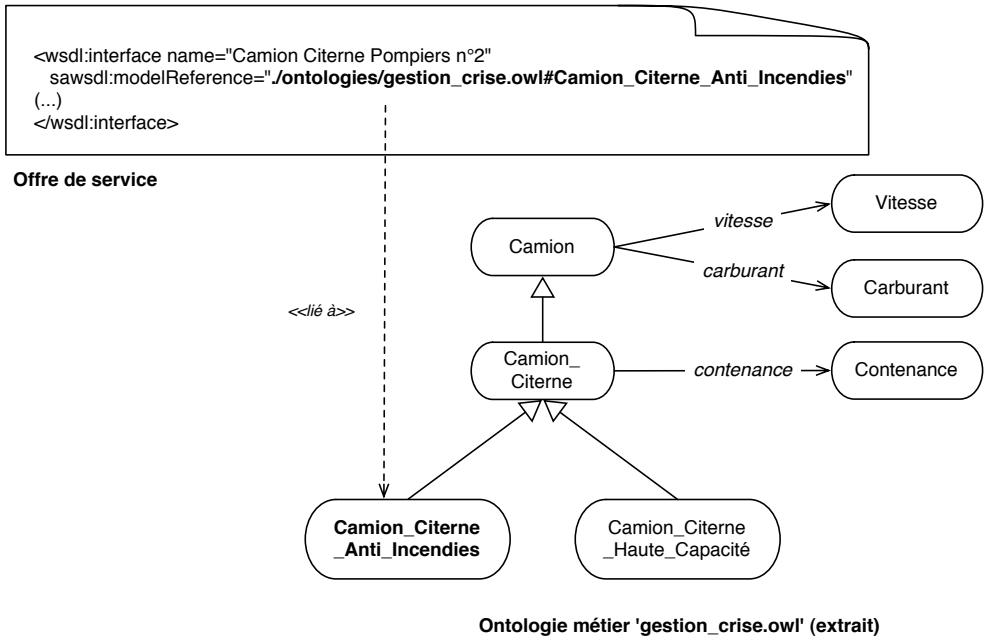

FIGURE 4.2 – Exemple d'annotation sémantique d'une offre spécifique de service.

Pseudo-code	**Méthode SAWSDL annotée**

aller_ a(emplacement) : status	`<wsdl:operation name="aller_a"`
	`sawsdl:modelReference=`
	`"./ontologies/gestion_crise.owl#aller_a">`
	`<wsdl:input element="emplacement"/>`
	`<wsdl:output element="status"/>`
	`</wsdl:operation>`

arroser(position, quantité) : status	`<wsdl:operation name="arroser"`
	`sawsdl:modelReference=`
	`"./ontologies/gestion_crise.owl#arroser">`
	`<wsdl:input element="position"/>`
	`<wsdl:input element="quantité"/>`
	`<wsdl:output element="status"/>`
	`</wsdl:operation>`

Le second volet de cette offre publique de service, qui va accompagner le fichier SAWSDL, concerne la description des propriétés non-fonctionnelles statiques du service, dont l'ensemble constitue alors un *contrat (de qualité) de service*. Dans le contexte du projet SemEUsE, ces informations sont couchées dans un document WS-Agreement offert par le service et stocké, avec le fichier SAWSDL, au sein d'un annuaire. La tableau suivant fait la liste exhaustive de ces propriétés.

Propriété non-fonctionnelle	**Contrat de Qualité de Service**
"Le camion sera capable de maintenir une vitesse comprise entre 150 et 200 Km/h"	$150 \leq$ vitesse (Km/h) ≤ 200
"Le camion disposera d'une réserve maximale en carburant de 500 Litres"	$0 \leq$ carburant (Litres) ≤ 500
"Le camion citerne aura une contenance maximale de 4000 Litres d'eau et une contenance minimale de 1000 Litres"	$1000 \leq$ contenance (Litres) ≤ 4000
"Le service camion sera capable de maintenir un temps de réponse strictement inféreur à 20 ms"	temps_de_reponse (ms) < 20
"Le temps de traitement pour la commande *arroser* ne sera pas supérieur à 10 ms"	arroser.temps_de_traitement (ms) ≤ 10

Drones de surveillance

Un second type de services présents à l'exécution sera celui des drones de surveillance mis à disposition par les différents services incendie, ainsi que par les militaires qui seront ici amenés à effectuer des opération conjointes avec les pompiers lors de la crise. On va considérer, lors ce cas d'utilisation, les offres statiques de services publiées par ces deux partenaires comme étant similaires fonctionnellement (*i.e.* mêmes interfaces) et très proches non-fonctionnellement. Les drones proposent alors de deux méthodes distinctes, l'une pour indiquer une zone à survoler, l'autre pour obtenir une image à partir de la caméra embarquée.

Pseudo-code	**Méthode SAWSDL annotée**
survoler_zone(emplacement) : status	`<wsdl:operation name="survoler_zone" sawsdl:modelReference= "./ontologies/gestion_crise.owl# survoler_zone"> <wsdl:input element="emplacement"/> <wsdl:output element="status"/> </wsdl:operation>`
obtenir_image() : image	`<wsdl:operation name="obtenir_image" sawsdl:modelReference= "./ontologies/gestion_crise.owl# obtenir_image"> <wsdl:output element="image"/> </wsdl:operation>`

On retrouve ensuite de nouveau le second volet non-fonctionnel des offres de service par la description de leurs propriétés statiques de Qualité de Service : elles portent en tout premier lieu sur la *sécurité* du canal de transmission de données avec le drone, la *bande-passante* offerte par le service et la *résolution* d'image disponible *via* la caméra embarquée. La correspondance non-fonctionnelle

des multiples offres statiques de service publiées par les pompiers et les militaires sur ces trois critères peut s'expliquer facilement par le fait que les drones dont ils disposent sont d'un modèle équivalent et proviennent du même fournisseur, il est donc logique qu'ils disposent de caractéristiques générales similaires. Cependant, comme il est d'usage entre les matériels civils et militaires, les bandes de fréquence utilisées pour leurs communications ne sont pas les mêmes. Le quatrième critère non-fonctionnel de *fréquence radio* sera donc différenciateur entre les drones des pompier et ceux des militaires, tel qu'indiqué ci-dessous. Néanmoins, une différenciation plus fine entre les services ne pourra s'effectuer que sur la base de leur QoS effective à l'exécution.

Propriété non-fonctionnelle	**Contrat de Qualité de Service**
"Le niveau de sécurité de communication (sur une échelle de 10) est compris entre 0 et 7"	$0 \leq$ sécurité ≤ 7
"La bande passante disponible sera maintenue à 50 Ko/s *a minima*"	bande passante (Ko/s) ≥ 50
"La résolution d'image sera comprise entre 6 et 10 Mpix "	$6 \leq$ résolution (Mpix) ≤ 10
Drone pompier : "La fréquence radio utilisée pour la communication sera comprise entre 2 et 2.5 Ghz "	$2 \leq$ fréquence radio (Ghz) ≤ 2.5
Drone militaire : "La fréquence radio utilisée pour la communication sera comprise entre 5 et 7 Ghz "	$5 \leq$ fréquence radio (Ghz) ≤ 7

Equipes au sol : pompiers et gendarmes

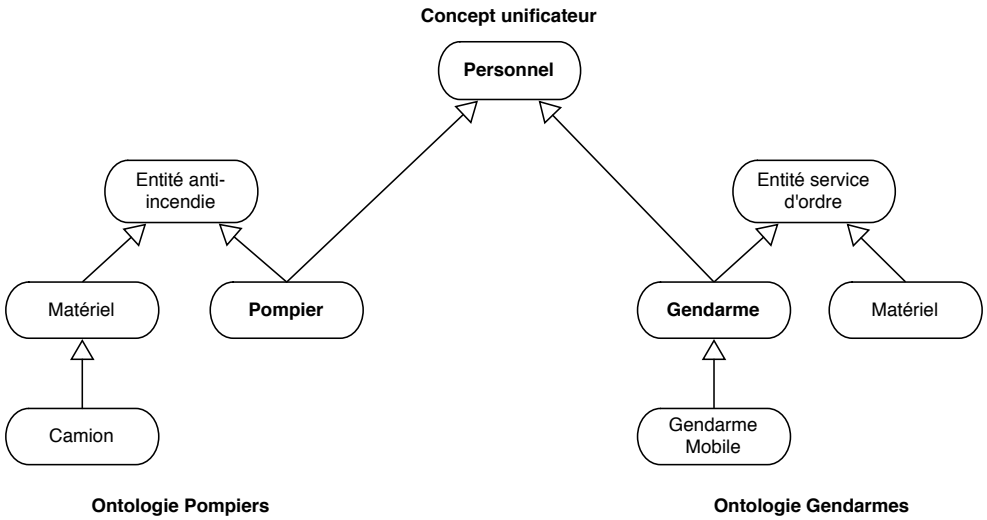

FIGURE 4.3 – Extrait d'ontologie couvrant les domaines Pompiers et Gendarmes.

Pour le dernier type de service en présence, on regroupe sous la notion de *Personnel*, concept unificateur défini dans une ontologie commune aux deux corps de métier (cf. figure 4.3), les pompiers et les gendarmes. Ils assureront en effet, dans le cadre précis de ce cas d'utilisation, le même type de

service métier : le déploiement au sol pour encercler la zone de l'incendie, à partir des informations précédemment obtenues grâce aux drones de surveillance. De ce fait, le concept fonctionnel *Personnel* permettra de désigner de manière unifiée et indirecte les pompiers et gendarmes dans les demandes de services. Ces demandes pourront ensuite être liées dynamiquement à des offres effectives de service annotées directement par le concept *Personnel*, ou par l'un de ses sous-concepts, lors de l'exécution des processus métiers.

Tout comme pour les drones que nous avons précédemment décrits, on considère en effet les offres en termes de services des pompiers et des gendarmes comme étant similaires sur le plan fonctionnel : elles sont respectivement annotées au niveau de leur interface SAWSDL par les sous-concepts *pompier* ou *gendarme* grâce à l'attribut *sawsdl :modelReference* et décrivent toutes la même méthode *encercler_ zone(emplacement, image)* ayant pour effet de communiquer un ordre de déploiement au personnel concerné, en lui fournissant l'emplacement de la zone ainsi que le relevé d'imagerie aérienne disponible.

Pseudo-code	**Méthode SAWSDL annotée**

encercler_ zone(emplacement, image) : status	`<wsdl:operation name="encercler_zone"`
	`  sawsdl:modelReference=`
	`  "./ontologies/gestion_crise.owl`
	`  #encercler_zone">`
	`  <wsdl:input element="emplacement"/>`
	`  <wsdl:input element="image"/>`
	`  <wsdl:output element="status"/>`
	`</wsdl:operation>`

En ce qui concerne la facette non-fonctionnelle des offres de service publiées dans le système, trois critères de QoS ont été retenus : il s'agit de la *résistance au feu*, de la *fatigue physique* et de la *fréquence radio* utilisée pour leurs communications qui, tout comme pour les drones, diffère entre civils (les pompiers) et militaires (les gendarmes). Les tableaux suivants représentent respectivement l'offre non-fonctionnelle de service générique utilisée par les pompiers et les gendarmes.

Propriété non-fonctionnelle pompier	**Contrat de Qualité de Service**
"La résistance au feu de l'équipement sera limitée à 120°C, mais il sera capable de supporter 50°C au minimum"	$50 \leq$ résistance au feu (°C) ≤ 120
"Le pompier sera capable de maintenir sa fatigue physique à 70 ou moins (sur une échelle de 100)"	fatigue physique ≤ 70
"La fréquence radio utilisée pour la communication sera comprise entre 2 et 2.5 Ghz "	$2 \leq$ fréquence radio (Ghz) ≤ 2.5

Propriété non-fonctionnelle gendarme	**Contrat de Qualité de Service**
"La résistance au feu de l'équipement sera limitée à 60°C, mais il sera capable de supporter 30°C au minimum"	30 $\leq$ résistance au feu (°C) $\leq$ 60
"Le gendarme sera capable de maintenir sa fatigue physique à 70 ou moins (sur une échelle de 100)"	fatigue physique $\leq$ 70
"La fréquence radio utilisée pour la communication sera comprise entre 5 et 7 Ghz "	5 $\leq$ fréquence radio (Ghz) $\leq$ 7

Une précision doit être apportée sur le mode de calcul de la *fatigue physique* telle que présentée dans les deux offres de service précédentes. En effet, cette dernière est en l'occurrence inférée à partir du *rythme cardiaque* et de la *respiration* des personnels qui, tout comme leur *résistance au feu*, peuvent être mesurées grâce à des capteurs présents sur leurs équipement. La formule utilisée pour son calcul est la suivante : $fatigue\ physique = a * rythme\ cardiaque + b * respiration$, où a et b sont deux variables d'ajustement qui permettent, en l'occurrence, d'obtenir une valeur finale comprise en 0 et 100.

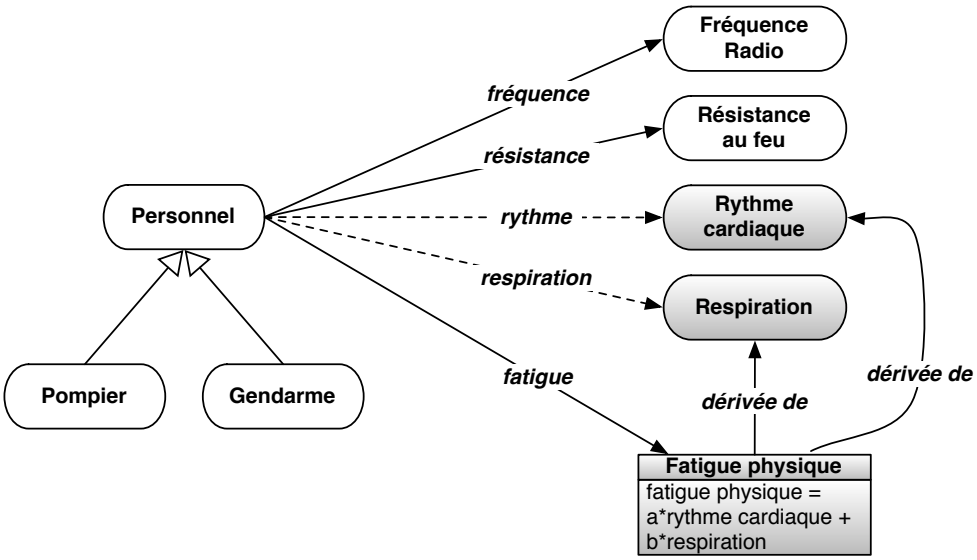

FIGURE 4.4 – Calcul de la fatigue physique à partir du rythme cardiaque et de la respiration.

Une solution possible pour le stockage de cette information consiste à décrire la *fatigue physique* et les autres critères non-fonctionnels comme des concepts ontologiques, au sein même de l'ontologie partagée par les pompiers et les gendarmes, et annoter les offres non-fonctionnelles de service en conséquence. La fatigue va alors être liée aux deux autres concepts de *rythme cardiaque* et de *respiration*, ainsi que fournir la formule mathématique nécessaire à son propre calcul (cf. figure 4.4).

Qu'il s'agisse des offres génériques de services de camion citerne d'intervention, de drone de surveillance, de pompier ou de gendarme que nous venons d'aborder, de nombreux services se conformant à ces offres devront être publiés pour qu'un besoin d'orchestration et sélection dynamique apparaisse

à l'exécution. De par l'ampleur de la crise traitée par ce cas d'utilisation, il s'agit bien du contexte en présence.

4.3.2 Scénario 2 : définition d'un processus abstrait de gestion d'incendie

Ce scénario consiste à établir statiquement, en amont de la survenue des incendies, un processus métier de résolution de crise sous une forme qui pourra être interprétée de manière automatique au moment opportun. En effet, même si l'on ne connaît pas au moment de l'écriture du processus les services qui seront seulement disponibles à l'exécution (mais qui restent pour autant nécessaires à son accomplissement), on est cependant déjà capable d'exprimer sans ambiguïtés ses besoins. Ainsi, grâce à l'utilisation ciblée d'ontologies métiers, et donc de sémantique, il est possible de diminuer significativement le fort couplage syntaxique entre le processus et les offres de service elles-mêmes

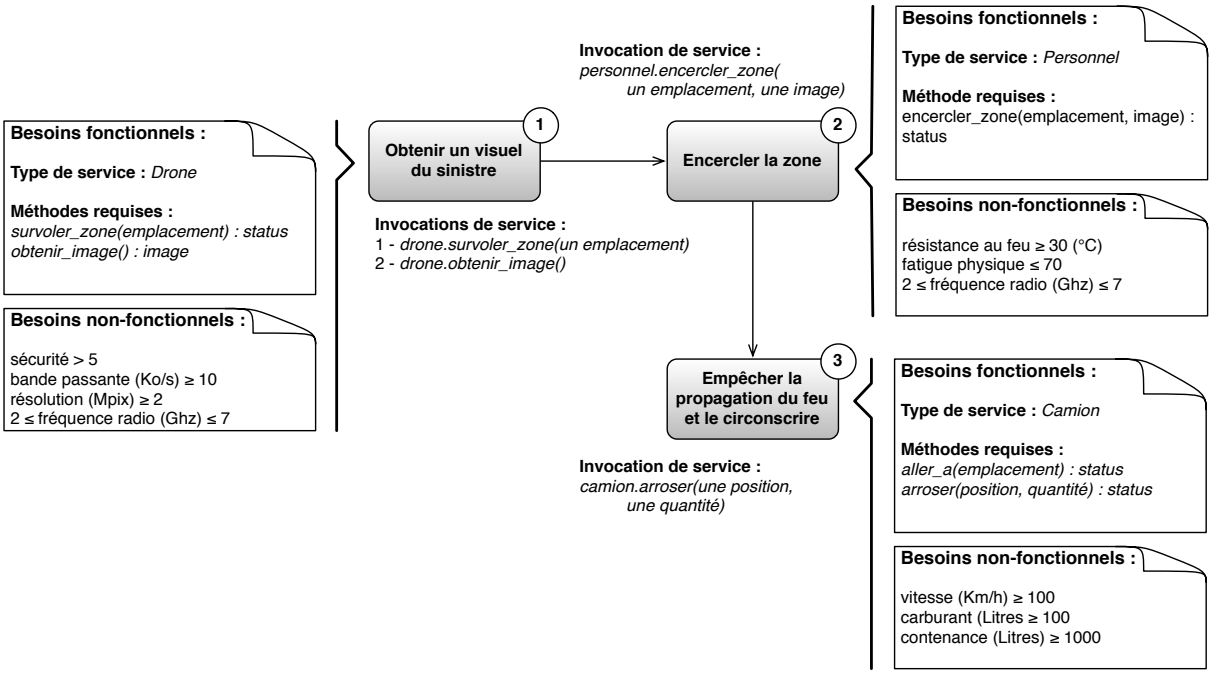

FIGURE 4.5 – Processus métier de gestion d'incendies.

La figure 4.5 présente un processus abstrait de gestion d'incendie qui pourra être déployé, sous la forme technique d'une description BPEL, dans le contexte d'intervention que nous avons précédemment introduit. Il décrit les actions de haut niveau suivantes à exécuter en séquence : *obtenir un visuel du sinistre*, *encercler la zone* et finalement, *empêcher la propagation du feu et le circonscrire*. A chacune de ces actions est associé un ensemble de besoins fonctionnels et non-fonctionnels établi statiquement, qu'il faudra faire correspondre, lors de l'orchestration de service, aux différentes offres disponibles.

Si l'on s'attarde sur la première action d'obtention d'images du théâtre des opérations, il faut noter que le type du service requis ainsi que ses méthodes sont indiqués sous la forme des besoins fonctionnels de l'action. Ils sont complétés par les besoins non-fonctionnels qui, sous l'apparence d'un document WS-Agreement requis lié au processus BPEL, fixent sous forme de bornes numériques les

exigences *nécessaires et suffisantes* au processus, et ceci sans connaissance préalable des offres de services qui seront disponibles lors de l'intervention. On remarque notamment une indifférence sur la bande de fréquence utilisée par les drones pour communiquer ($2 \leq$ *fréquence radio (Ghz)* ≤ 7), ce qui signifie que l'on est capable d'utiliser sans distinction les services de types civils ou militaires.

Dans le cas de la seconde action, on constate que, d'un point de vue strictement fonctionnel, tous les pompiers et gendarmes disponibles seront requis pour l'intervention : cela se traduit par l'utilisation du concept ontologique général *Personnel* pour désigner les deux types de services au lieu de leurs concepts métiers spécifiques. Par ailleurs, les besoins non-fonctionnels sont suffisamment lâches pour que tous les services respectant le modèle générique d'offres statiques de service introduit dans la sous-section précédente soient considérés comme conformes. Cependant, si la stricte conformité fonctionnelle et non-fonctionnelle des services orchestrés est considérée comme un préalable indispensable à leur utilisation, une information supplémentaire sera alors forcément nécessaire au niveau du processus BPEL si l'on souhaite être capable de discriminer, à l'exécution, entre ces services conformes dans le but d'en prélever le ou les plus utiles à la résolution de la crise.

En ce qui concerne la troisième action, on remarque que seuls les camions anti-incendies sont sollicités pour empêcher la propagation du feu et le circonscrire, et non les personnels au sol, cet état de fait est du au type d'incendie visé : ce processus a été établi pour traiter des feux à fort dispersement géographique qui induisent de multiples déplacements sur de grandes distances nécessitant l'utilisation de véhicules adaptés. Les besoins non-fonctionnels pour réaliser cette action sont à l'avenant : une forte contenance en eau ($\geq$ 1000 Litres) et une vitesse élevée ($\geq$ 100 Km/h).

4.3.3 Scénario 3 : élicitation de préférences utilisateur spécifiques à la crise

Le processus métier que nous venons de détailler a été défini avant que la crise bien spécifique, qui nous intéresse dans ce cas d'utilisation, soit avérée. De fait, chaque incendie est unique et il peut être nécessaire, dans les instants qui précèdent le déploiement du processus, de compléter les besoins non-fonctionnels établis statiquement par des préférences utilisateur spécifiquement conçues pour les besoins propres à ce type d'incendie de particulièrement grande ampleur. Les services en présence lors de l'exécution pourront par la suite être départagés sur la base de leur adhésion à ces préférences (et donc de leurs *utilités* respectives).

Le formalisme dédié à la modélisation de préférences utilisateur que nous avons défini dans le cadre de cette thèse sera présenté en détails au chapitre 6, il permet la modélisation de déclarations qualitatives et l'utilisation de termes linguistique au sein d'un modèle graphique. Nous nous contenterons donc ici d'une élicitation abstraite de ces préférences.

En ce qui concerne la première action d'obtention d'un visuel du sinistre grâce aux drones, les préférences suivantes sont établies :

- *"On souhaite, dans l'absolu, utiliser les drones disposant, pour la transmission des images, des meilleures valeurs en termes de sécurité, bande passante et résolution."*
- *"On préfère cependant optimiser la bande passante au détriment de la sécurité ou de la résolution"*, qui sont des critères de moindre importance dans ce contexte non militaire.
- *"On effectue par ailleurs un compromis entre l'optimisation de la bande passante et celle de la résolution : si la bande passante est faible, alors on favorisera les services disposants d'une faible*

résolution d'image, et inversement." L'objectif étant dans ce contexte d'obtenir les images le plus vite possible en minimisant la latence.

A la deuxième action mettant à l' œuvre les personnels présents au sol pour encercler la zone de l'incendie, sont associées les préférences suivantes :

- *"On souhaite, dans l'absolu, utiliser les personnels disposant de la meilleure résistance au feu et d'une fatigue physique minimale."*
- *"De par l'intensité de l'incendie, on favorise la résistance au feu au détriment de la fatigue physique"*, de plus, ce sont les camions qui vont effectuer l'essentiel des déplacements au sol, l'intervention n'en sera donc que moins exigeante sur l'état physique des personnels.

Pour finir, concernant la troisième et dernière action utilisant les services de camion citerne pour empêcher la propagation du feu et le circonscrire, les préférences portent tout particulièrement sur leurs caractéristiques métiers de *contenance* et de *vitesse* :

- *"On souhaite obtenir les meilleurs valeurs de vitesse et de réserve de carburant."*
- *"De par l'étendue de l'incendie, on favorise la vitesse de déplacement et la réserve en carburant effective des camions au détriment de leur contenance en eau"*, et ceci de manière à ce qu'ils puissent se déplacer le plus rapidement possible et avec la plus grande autonomie.
- Toujours dans le même soucis de rapidité de déplacement pour maximiser la couverture de l'incendie, *"on préfère une contenance moyenne (plutôt qu'élevée) des camions en eau, de manière à les alléger. Cependant, pour qu'ils restent utiles, on écarte autant que possible ceux qui n'auraient qu'une faible contenance"*.

Ces préférences s'inscrivent *à l'intérieur* des bornes de QoS précédemment définies. Il s'agit donc bien de départager, à l'exécution, des services concurrents, mais *conformes* fonctionnellement et non-fonctionnellement aux besoins du processus métier. Suite à leur élicitation, ces préférences sont accolées aux différents sites d'appels de services du processus (cf. figure 4.6).

4.3.4 Scénario 4 : composition agile de services pendant la crise

Suite au déclenchement des alertes, des actions doivent être entreprises immédiatement, et ce à tous les niveaux. Le CODIS [12] des Alpes Maritimes a alors pour mission d'assurer la composition de l'intervention au niveau départemental.

En se rapprochant du théâtre des opérations, cette composition va se décliner à des niveaux de détail de plus en plus fins. On se focalise alors dans ce scénario sur le déclenchement du processus métier que nous avons défini précédemment, dans le cadre d'une sphère opérationnelle faisant notamment intervenir plusieurs drones avec caméras embarquées et équipes au sol (cf. figure 4.7).

Le déploiement rapide des équipes ainsi que la présence diversifiée de matériel va nécessiter en tout premier lieu la mise en place d'une composition active des services disponibles, par un mécanisme de *liaison tardive*. Cette approche, rendue possible par le couplage lâche implanté entre les offres et demandes de service que nous avons decrites dans les *scenarii* 1 et 2, est détaillée au chapitre 5.

12. Les CODIS sont les Centres Opérationnels Départementaux d'Incendie et de Secours.

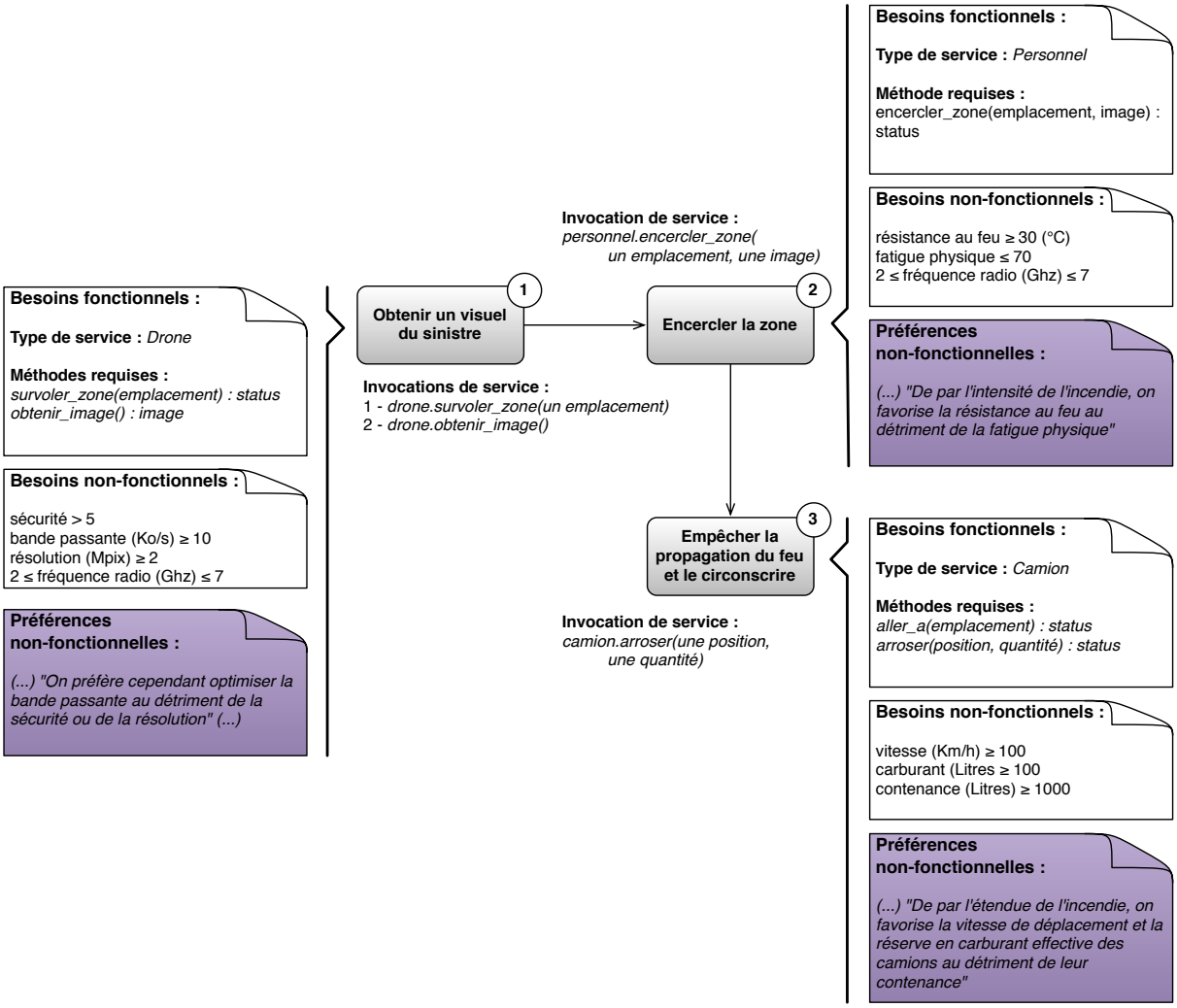

FIGURE 4.6 – Processus métier de gestion d'incendies spécialisé par l'utilisation de préférences.

Par ailleurs, pour maximiser la pertinence et la performance du processus métier de gestion d'incendies dans ce contexte spécifique, la Qualité de Service courante des drones, des personnels au sol et des camions sera intégrée, au cœur de la composition de services, par une approche *utile* exploitant les préférences définies par l'utilisateur. Ce second aspect de la composition sera détaillé dans le chapitre 6 en prenant notamment pour exemple la sélection d'un drone d'imagerie à partir des préférences définies dans le troisième scénario.

4.4 Conclusion

Dans ce chapitre, nous avons posé les bases d'un cas d'utilisation simplifié en gestion de crise environnementale, dans le cadre des feux de forêt de grande envergure. Il permet, décliné en quatre *scenarii* distincts, d'illustrer les différentes facettes de nos contributions scientifiques sur la composi-

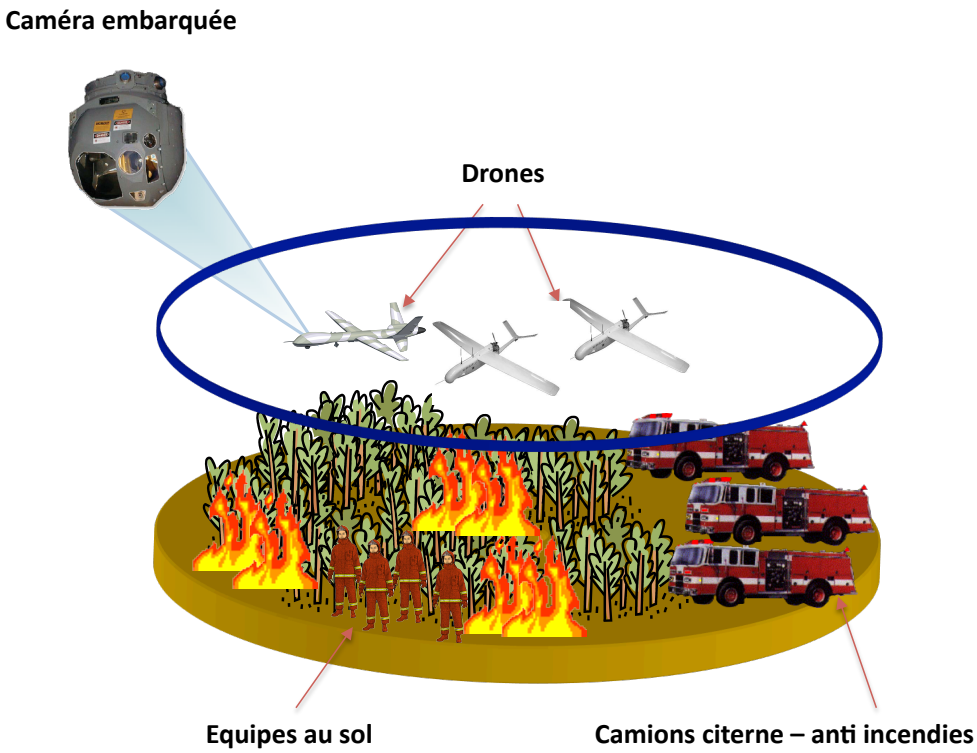

FIGURE 4.7 – Cas d'utilisation : sphère opérationnelle

tion de services, mais donne aussi des détails sur les phases préalables à la composition : la définition d'offres de services et processus métiers en couplage lâche, ainsi que celle de préférences utilisateur qui seront exploitées à l'exécution du système. Ce cas d'utilisation montre qu'il est ainsi possible de tisser des liens étroits entre les problématiques usuelles de la gestion de crise et nos travaux sur les architectures SSOA. Le contexte métier qui a été ici défini sera repris, au besoin, dans le reste de ce manuscrit pour illustrer nos propos.

Deuxième partie

Contributions et mise en œuvre

Chapitre 5

Composition active de services

Sommaire

5.1 Introduction

Dans le contexte de l'implantation d'applications réparties à large échelle, tout particulièrement *via* le Web, les architectures de type SOA sont devenues, au cours des ans, un élément de réponse incontournable aux problématiques de répartition et de communication entre les entités de ces systèmes complexes. Comme nous l'avons vu, une seconde itération de ces architectures, les SSOA, vient compléter sans rupture technologique franche le panel des facilités mises à la disposition des fournisseurs de services : un meilleur découplage entre offres et demandes de services, ainsi qu'une augmentation du niveau d'abstraction lors de leur spécification.

Bien que l'approche SSOA pose les bases indispensables à notre travail, elle reste cependant muette quant aux problématiques levées par le contexte dans lequel s'inscrit ce manuscrit. Dans ce dernier, les services utilisés par les applications réparties sont rarement connus au moment de l'écriture des processus métier, il est par ailleurs nécessaire de faire preuve d'une grande agilité à l'exécution de manière à pouvoir incorporer tardivement, au processus de composition, des services découverts dynamiquement, tout en prenant en compte de multiples contraintes non-fonctionnelles.

Bien que ces problématiques découlent ici directement de notre contexte d'application (elle sont mises en exergue par notre cas d'utilisation en gestion de crise environnementale détaillé au chapitre 4), on peut aisément concevoir leur pertinence au delà de ces frontières.

Ainsi, la nécessité de répondre à ces diverses questions se traduit, dans le cadre de nos travaux, par la mise en œuvre d'une *composition active de services via* la description et l'implantation d'une *approche pour la liaison tardive de services* lors de leur orchestration. Sont alors définis dans ce chapitre différents concepts et abstractions de programmation à adopter pour implanter une *décision de liaison* tardive et efficace des fournisseurs aux consommateurs de services. De même, nous mettons en avant l'importance des décisions de liaison qui ne peuvent (ou ne doivent) être prisent que lors de l'exécution des processus, sur la base de leurs besoins non-fonctionnels et des caractéristiques courantes des services disponibles. L'approche permettant d'effectuer ces décisions de liaison, ainsi que le choix du moment le plus opportun pour les réaliser, constituent dès lors le cœur de notre composition active de services.

Ce chapitre est organisé comme suit : en tout premier lieu, on présente en 5.2 une comparaison entre les compositions actives et dynamique. Ensuite, en 5.3, on avance les principes dirigeant notre approche pour la liaison tardive de services lors de leur orchestration. On y aborde notamment les bénéfices à tirer de cette approche dans le contexte des SSOA. Les sous-chapitres 5.4 et 5.5 décrivent ensuite les étapes préliminaires qui sont nécessaires à la mise en œuvre de la liaison tardive à proprement parler, elle même approfondie dans le sous-chapitre 5.6.

5.2 Composition active *vs.* composition dynamique

Dans le cadre de l'état de l'art de ce manuscrit (cf. chapitre 3), nous avons abordé de multiples travaux pré-existant dans le domaine de la composition de services sous contraintes non-fonctionnelles. Ils sont présentés comme "dynamiques" dans la mesure où ils sont capables d'effectuer, lors du déploiement des processus, un ensemble de choix de liaison, en fonction des contraintes non-fonctionnelles des processus et des propriétés contractuelles statiques des services.

Ces choix d'affectation étant effectués en majeure partie lors du chargement des processus dans le canevas d'exécution, c'est-à-dire avant l'exécution de leur partie métier, ils ne sont remis en cause par recomposition que lors de *violations manifestes des contraintes non-fonctionnelles* sur lesquelles se sont accordés les différents acteurs de la composition. On peut comparer ce mode de fonctionnement à celui d'un mécanisme de gestion d'"exceptions".

Ainsi, en reprenant après composition l'exemple que nous avions introduit à la section 3.3.1 (cf. figure 5.1) ; si pendant l'exécution de *process* le service S_2 ne respecte pas son contrat et dépasse 20ms ($t_{S2} = 30ms$), tout le reste de la composition est remis en question car, dans le pire des cas, $t_{process} = 43ms$. Il est alors nécessaire de gérer cette exception en effectuant une recomposition du segment en cours d'exécution de *process*, ou de le recomposer en totalité (ici *Affectation #2*).

On peut donc qualifier ces approches pour la composition dynamique de services comme étant *"défensives"*. La recomposition des services, processus particulièrement coûteux en temps malgré les optimisations, y est présentée bien souvent comme l'unique moyen d'adaptation aux contraintes environnementales courantes, et n'est déclenchée qu'*en réaction à une erreur patente* pendant l'exécution du processus.

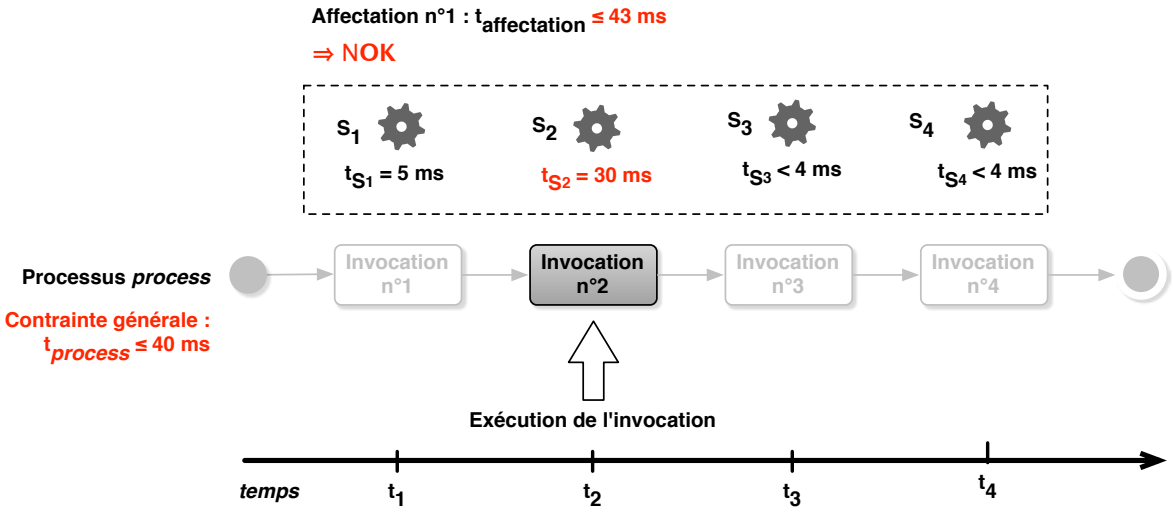

FIGURE 5.1 – Composition dynamique : violation de contrainte à l'exécution.

Nous proposons alors un nouveau mécanisme qui peut être considéré comme *complémentaire* de cette composition dynamique défensive : la *composition active de services*. Nous nous démarquons alors des approches traditionnelles dans la mesure où la recomposition n'y est considérée que comme dernier rempart aux écarts les plus importants lors de l'exécution. Sans attendre son intervention, la composition active va sélectionner *pendant* l'orchestration de services, pour chaque site d'invocation, le service répondant le mieux, *à cet instant*, aux contraintes non-fonctionnelles locales, par l'intégration de ses propriétés courantes mesurées (cf. figure 5.2). Ce mécanisme de sélection fait partie intégrante de notre approche pour la *liaison tardive de services* présentée ci dessous.

La sélection d'un service, qui est locale à chaque site d'invocation, est effectuée *au sein d'un groupe de services fonctionnellement équivalents*. Ces groupes sont prédéterminés par le biais d'un mécanisme de filtrage que nous abordons dans la section 5.4. Il pourraient aussi être le fruit d'un processus amont de composition dynamique : si les affectations globales de services issues de la composition dynamique ne prévoient normalement qu'un seul service pour chaque site d'appel (cf. affectation n°1 sur le figure 5.1), il est cependant raisonnable d'avancer que la composition dynamique serait aussi en mesure de fournir plusieurs alternatives pour chaque sites d'appels. Il s'agit là d'une nouvelle marque de la complémentarité potentielle entre approches dynamiques et composition active.

5.3 Une approche pour la liaison tardive de services

Si la prise en compte des contraintes de Qualité de Service lors de l'exécution des processus métier apparaît effectivement comme importante, elle ne doit pas pâtir, dans notre contexte SSOA, du faible couplage entre processus et services (cf. section 2.1.3) : *nous avançons qu'elle peut même en bénéficier* si elle est correctement exploitée. En effet, ce couplage lâche, qui s'exprime lors de la spécification d'un processus métier, induit tôt *ou tard* la nécessité d'une décision de liaison effective à un service concret de manière à pouvoir poursuivre son exécution. Nous constatons alors que la

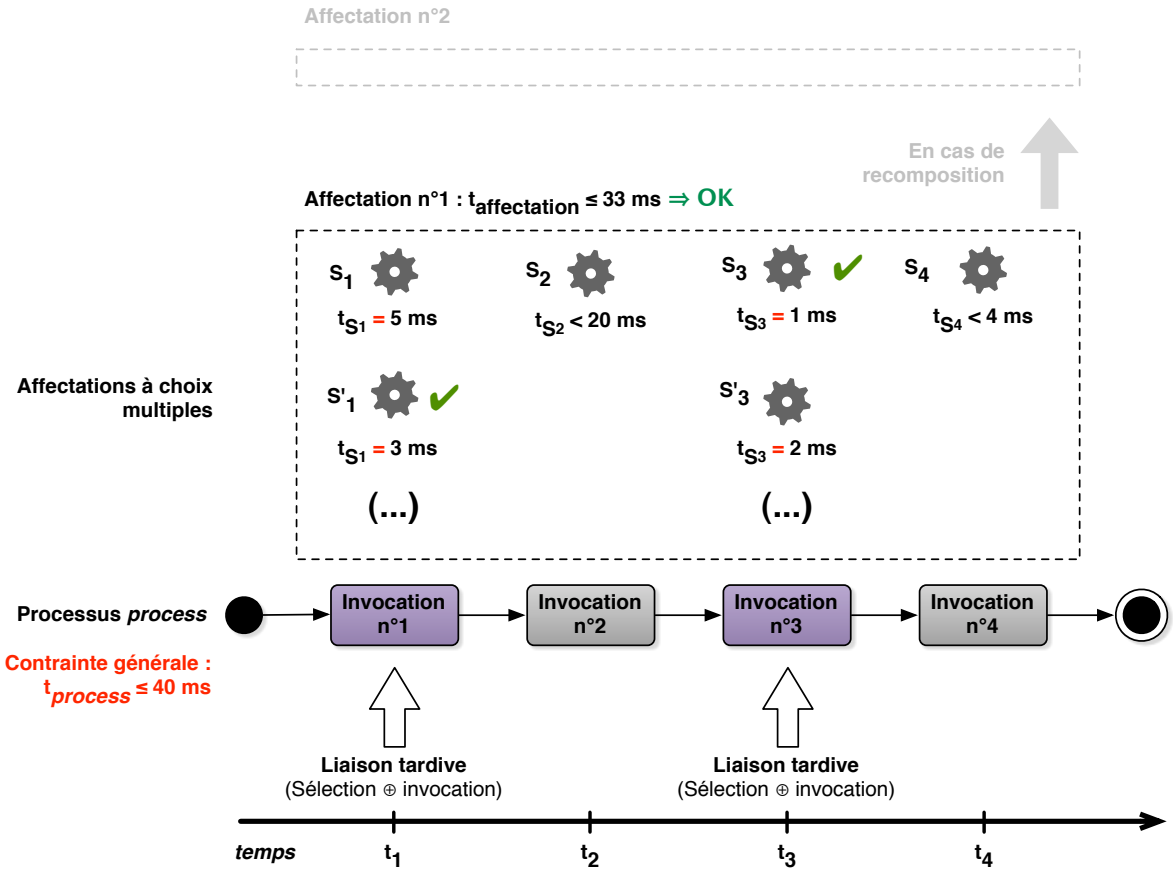

FIGURE 5.2 – Composition active de services.

pertinence de cette décision de liaison, effectuée notamment sur la base de critères non-fonctionnels, serait d'autant plus grande qu'elle reposerait sur des informations de QoS ponctuelles, recueillies *le plus tardivement possible*.

Par conséquent, nous avançons que la décision doit être prise à la volée, à l'extremum du cycle de vie d'un processus : en l'occurrence pendant son orchestration, lors de chaque appel de service qui en émane. Ce positionnement temporel disqualifie *per se* les liaisons fermement établies au moment de la définition du processus client ; leur pertinence non-fonctionnelle étant relativement très faible, si ce n'est inadéquate. De cette observation naît notre démarche de liaison tardive de services [Châtel et al., 2010a], tout particulièrement justifiée par les variations fréquentes du niveau courant de QoS des services ainsi que la durée moyenne d'exécution relativement longue des processus métiers dans le contexte du cas d'utilisation en gestion de crise environnementale que nous avons introduit. La liaison tardive des services de *drones*, *pompiers* et *camions* au processus métier de gestion d'incendie que nous avons précédemment introduit est illustrée par la figure 5.3 : à chaque activité est associée une décision de liaison effectuée sur la base des valeurs de QoS recueillies tardivement sur les équipements et personnels disponibles.

Cependant, l'impact de cette démarche sur les performances des intergiciels de support des SSOA ne doit pas être négligé. En effet, tandis que cette nouvelle forme de liaison ouvre la voie

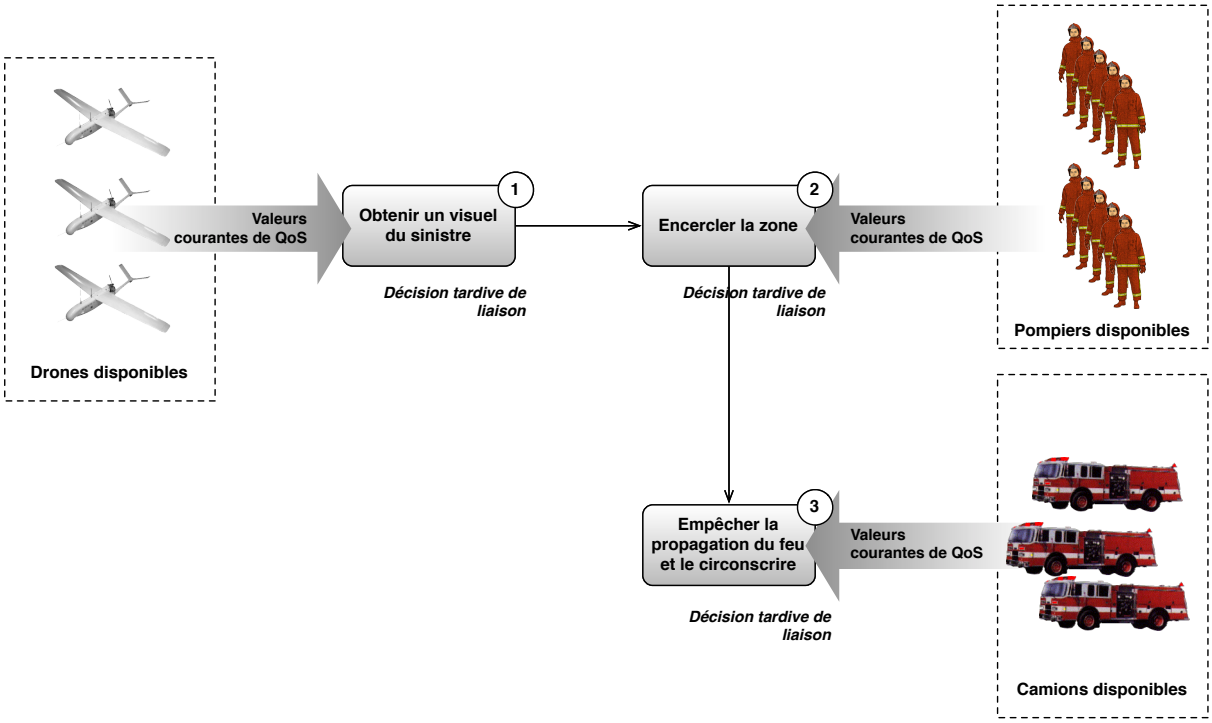

FIGURE 5.3 – Cas d'utilisation et liaison tardive.

à des applications réparties plus agiles et plus robustes, les problématiques et spécificités liées à son implantation doivent être étudiées et traitées. Pour minimiser cet impact, nous proposons la mise en place d'une approche globale pour la liaison tardive de services. Cette approche inclue deux étapes préliminaires à la phase ultime de liaison tardive à proprement parler ; de manière à passer outre, au moment de de la décision de liaison et de l'appel de service, certains délais liés aux premiers algorithmes d'appariement et au recueil des informations nécessaires à la décision. Pour ce faire, nous proposons premièrement de préétablir *par filtrage statique* un ensemble de services candidats par source d'appel dans les processus métiers, avant leur exécution ; ces ensembles seront alors immédiatement *supervisés* de manière à pouvoir obtenir à tout moment l'état non-fonctionnel courant de chaque service lors de l'exécution des processus. Reste alors à effectuer, au moment précis de l'appel de service, les *décisions tardives de liaison* qui s'imposent, en considérant ces ensembles prédéterminés de services et leurs valeurs courantes, pertinentes, de QoS.

Nous nous focalisons donc, dans les sections suivantes, sur les diverses étapes logiques propres à cette approche de liaison tardive de services. Elles sont présentées de manière condensée dans la figure 5.4 et dénombrées ci-après :

1. en tout premier lieu, le filtrage des services, détaillé en 5.4, ce dernier étant effectué au niveau des annuaires sur critères fonctionnels et non-fonctionnels. C'est à l'issue de cette étape que les ensembles de services candidats sont obtenus. Elle englobe par ailleurs, le cas échéant, le calcul des modèles d'adaptation de données utilisés lors du futur dialogue entre les services retenus et le processus métier client de ces services ;

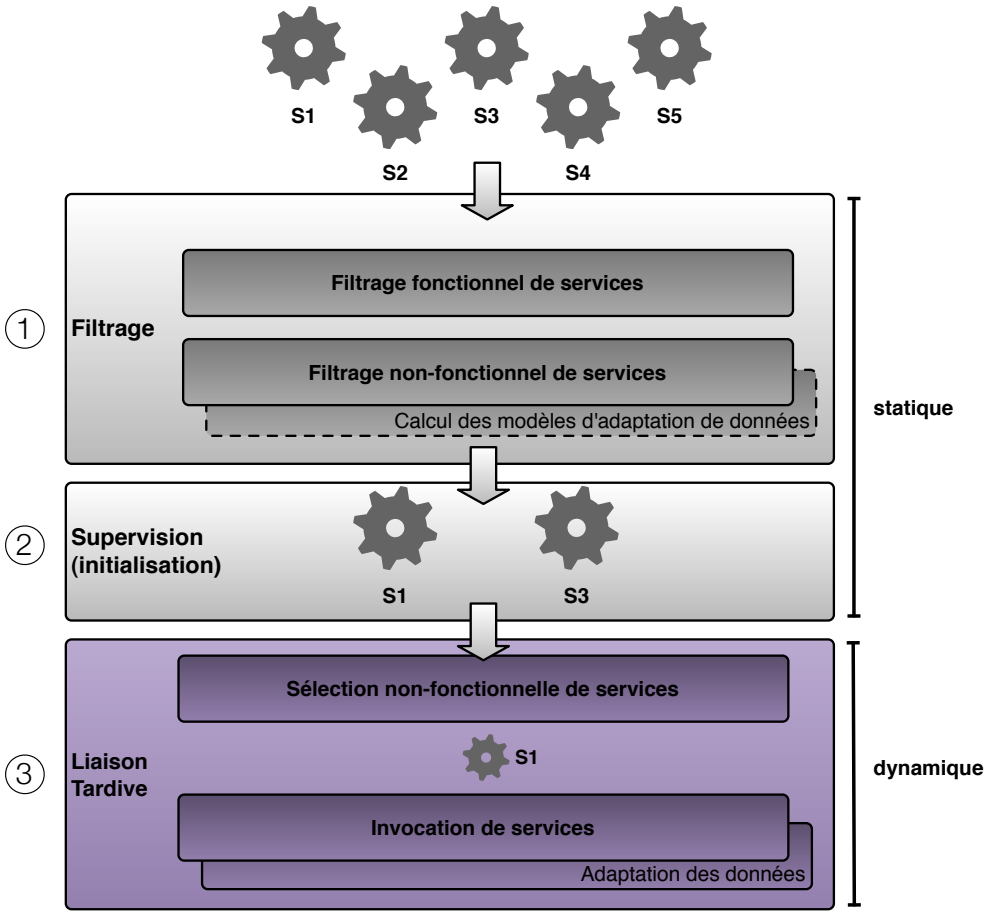

FIGURE 5.4 – Etapes d'une approche pour la liaison tardive de services.

2. suivi ensuite par la mise en place et le déclenchement de la supervision de services à partir de ces ensembles, détaillée en 5.5. Cette supervision sera effectuée en parallèle de l'exécution des processus métiers, de manière à pouvoir obtenir des informations "fraîches" sur l'état courant des services, et ce à tout moment ;

3. et pour finir, la liaison tardive d'un service, vue en 5.6 qui se décompose en une première étape de sélection du service considéré comme le plus approprié à cet instant, puis une seconde étape d'invocation de ce service adaptation des données échangées. La décision de liaison faisant notamment appel aux valeurs de QoS obtenues *via* le canevas de supervision des services préalablement déployé.

On remarque que l'on retrouve respectivement dans ces étapes logiques les deux niveaux logiciels abstraits de *filtrage des services et exécution du processus métier* et *contrôle et décision* évoqués précédemment (cf. section 1.4).

5.4 Filtrage de services

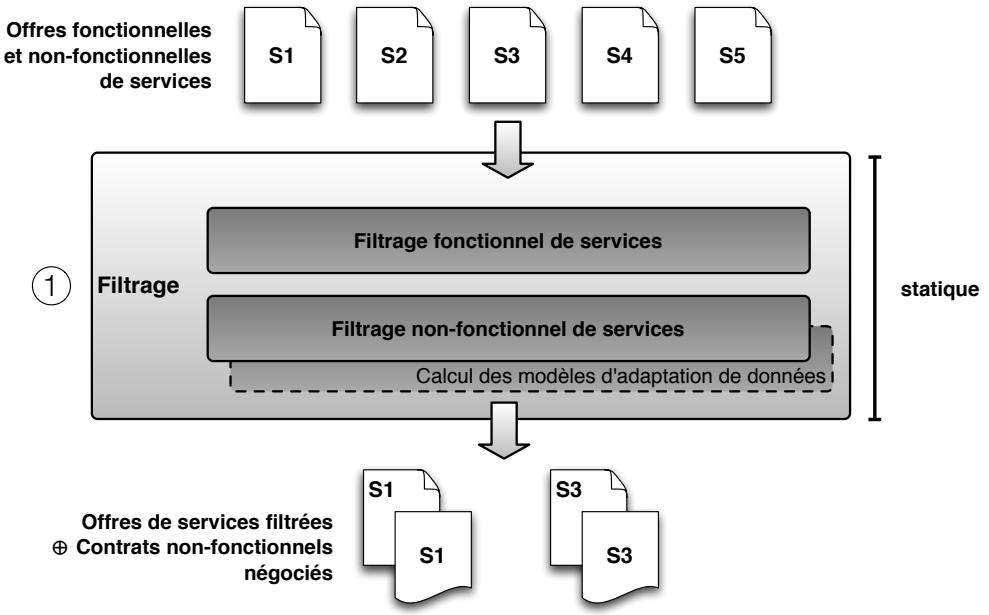

FIGURE 5.5 – Filtrage de services.

Dans notre contexte de SOA Sémantique, les offres fonctionnelles et non-fonctionnelles de services vont être publiées et stockées dans des annuaires prévus à cet effet, de manière à pouvoir être "découvertes" et associées à des processus métiers, le moment venu. Les annuaires de services sont alors tout désignés pour implanter cette fonctionnalité de filtrage des services disponibles dépeinte par la figure 5.5. Ainsi, à partir des besoins fonctionnels et non-fonctionnels (syntaxiques et sémantiques) qui lui sont communiqués pour chaque source d'appel de service d'un processus métier, un annuaire doit alors être capable d'établir l'ensemble des services pertinents et communiquer ce résultat. Ce procédé, qui implique notamment un raisonnement sur des concepts ontologiques, a un coût temporel non-négligeable, proportionnel au nombre de services répertoriés par l'annuaire et à la complexité de la requête.

5.4.1 Impact du calcul des modèles pour l'adaptation de données

Le filtrage qui permet la construction d'ensembles de services candidats doit être complété, tôt ou tard, par le calcul de modèles d'adaptation entre les structures de données tels qu'utilisées lors de la définition des besoins des processus et ceux exposés dans les offres concrètes des services candidats, quitte à éliminer certains services retenus si le l'obtention de ces modèles d'adaptation est impossible. En effet, dans les SSOA, à partir du moment où l'appariement est dirigé par des critères sémantiques de haut niveau, rien ne garantit une parfaite correspondance entre processus et services au plus bas niveau syntaxique ; niveau qui concerne les structures de données utilisées lors du passage d'arguments aux services et la récupération de leurs valeurs de retour. Une première approche de traitement de cette problématique d'adaptation de données a été précédemment abordée dans le

contexte des service Web sémantiques (cf. 2.1.3), elle consiste à utiliser les capacités d'annotation de la spécification SAWSDL pour lier par transitivité offres et besoins au travers d'une ontologie pivot, *via* la définition de transformations de données au format XML. L'ensemble de ces transformations constitue alors le modèle d'adaptation des données à appliquer lors de l'exécution d'une application Web. Cette solution technique ne précise cependant pas à quel moment les transformations doivent être calculées ; et il ne faut pas sous-estimer, là non plus, le coût de l'obtention de ces transformations, qu'elle soit manuelle ou effectuée par inférence logique.

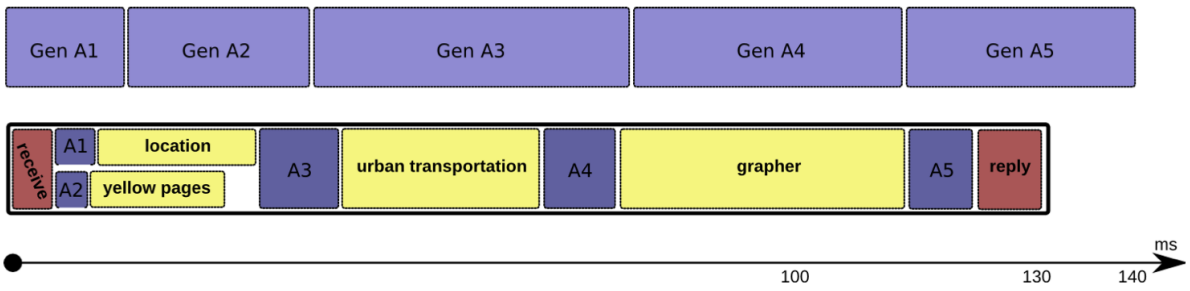

FIGURE 5.6 – Comparaison des temps d'exécution de la génération des adaptateurs et de l'exécution de la composition.

Afin d'illustrer l'impact très concret du calcul des modèles d'adaptation sur l'étape de filtrage des services, nous prenons pour exemple les récents travaux de Moreau *et al.* [Moreau et al., 2009] sur "la mise en œuvre automatique de processus métier dans le domaine des architectures orientée services". Ils cherchent à mettre au point de nouvelles solutions pour l'adaptation (automatique) de données et la sélection de services. Il s'agit donc bien ici d'obtenir des modèles d'adaptation par une inférence logique élaborée, sans intervention humaine. La génération automatique d'"adaptateur" est effectuée par une étude particulièrement poussée de la structure des schémas de données à apparier, dirigée par des informations de niveau sémantique disposées de part et d'autre. Cependant, l'étude relative des performances du générateur d'adaptateur sur un cas d'utilisation concret montre clairement les limites de cette approche pour une utilisation dynamique au cours de l'orchestration des services : le cas d'utilisation UTP (où "Urban Trip Planner") défini par Baligand *et al.* [Baligand et al., 2007] et repris par Moreau *et al.* propose un processus métier où cinq activités d'invocation de services externes nécessitent la génération d'autant d'adaptateurs de données. Si l'on compare sur 150 itérations le temps d'exécution moyen de la génération consécutive de ces cinq adaptateurs sous forme de feuilles XSL (au total *140 ms*), avec celui de la composition elle-même (*130 ms*), on constate une quasi-équivalence des deux durées ; ce qui signifie que si les deux processus (génération des adaptateurs et composition de services) sont effectués conjointement, le temps d'exécution total du processus métier *va plus que doubler*. La figure 5.6 compare ainsi les résultats obtenus pour la génération d'adaptation au temps d'exécution des différentes tâches de l'orchestration. La longueur des pavés est proportionnelle à leur temps d'exécution en milliseconde. Les pavés $GenA_i$ représentent les temps d'exécution du module de génération d'adaptateurs pour les activités A_i correspondantes.

Nous en déduisons que le filtrage des services disponibles ainsi que le calcul des modèles d'adaptation de données doivent être effectués *préalablement à l'exécution du processus métier*, et ce pour d'évidentes raisons de performance : par un tel procédé on minimise l'impact sur l'exécution du pro-

cessus des algorithmes d'appariement de services, de calcul des modèles d'adaptation de données, et des accès réseau subséquents.

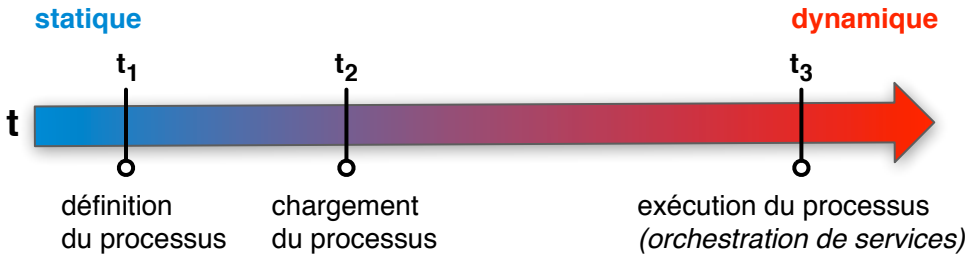

FIGURE 5.7 – Transition progressive du statique au dynamique.

Cependant, cette approche d'application précoce des algorithmes se heurte aux caractéristiques intrinsèques du contexte, et plus particulièrement à celle de couplage lâche des services aux processus qui plaide, au contraire, pour leur mise en œuvre tardive. Un "juste milieu" doit donc être trouvé entre ces deux tendances antagonistes, il passe par une étude plus approfondie du cycle de vie des processus. De fait, on distingue, dans le cadre d'un SSOA, deux principaux "moments" auxquels peuvent se rattacher les différentes étapes qui doivent précéder l'exécution d'un processus métier : lors de la définition du processus (*t1*), ou bien au moment du chargement préalable à son exécution (*t2*). Dans ces deux cas, on parlera alors d'une étape *statique* puisqu'effectuée avant l'exécution du code métier de l'application répartie ; par opposition à une étape dite *dynamique* qui pourra être mise en œuvre "juste à temps" ("just-in-time") au cours de l'exécution d'un processus (*t3*). Il faut cependant rester conscient qu'il n'existe pas de stricte séparation entre les moments statiques et dynamiques, plutôt une transition progressive, telle qu'illustrée par la figure 5.7 où l'on retrouve les moments de définition, chargement et exécution d'un processus.

5.4.2 Options pour la répartition temporelle des tâches de filtrage et de calcul des modèles

En se basant sur le cycle de vie des processus métiers, la figure 5.8 présente de multiples solutions (numérotées de 1 à 4) quant à la répartition des tâches de filtrage et de calcul des modèles d'adaptation de données. Le choix d'une option par rapport à l'autre s'effectue lors de l'implantation de l'architecture de support de la composition active de services, en fonction des contraintes liées à son contexte usuel de déploiement.

– La première option de filtrage et calcul statique des modèles d'adaptation a été suivie par Moreau *et al.* [Moreau et al., 2009]. Ces travaux mettant en œuvre des techniques élaborées pour le calcul des modèles d'adaptation de données entre offres et besoins de services, les algorithmes incriminés sont particulièrement gourmands en temps, comme il a été précédemment évoqué. Cependant, puisque le filtrage des services et le calcul des modèles d'adaptation de données sont ici effectués tous deux en *t1*, cette option présente l'avantage de lever en très grande partie les contraintes temporelles sur leur exécution. Par conséquent, il est donc possible de procéder à des appariements plus fins, ou qui seraient peut-être même inaccessibles à des algorithmes plus simples et donc moins coûteux en temps. Toutefois, le couplage lâche entre services et processus dont nous somme tributaires proscrit cette approche dans la mesure

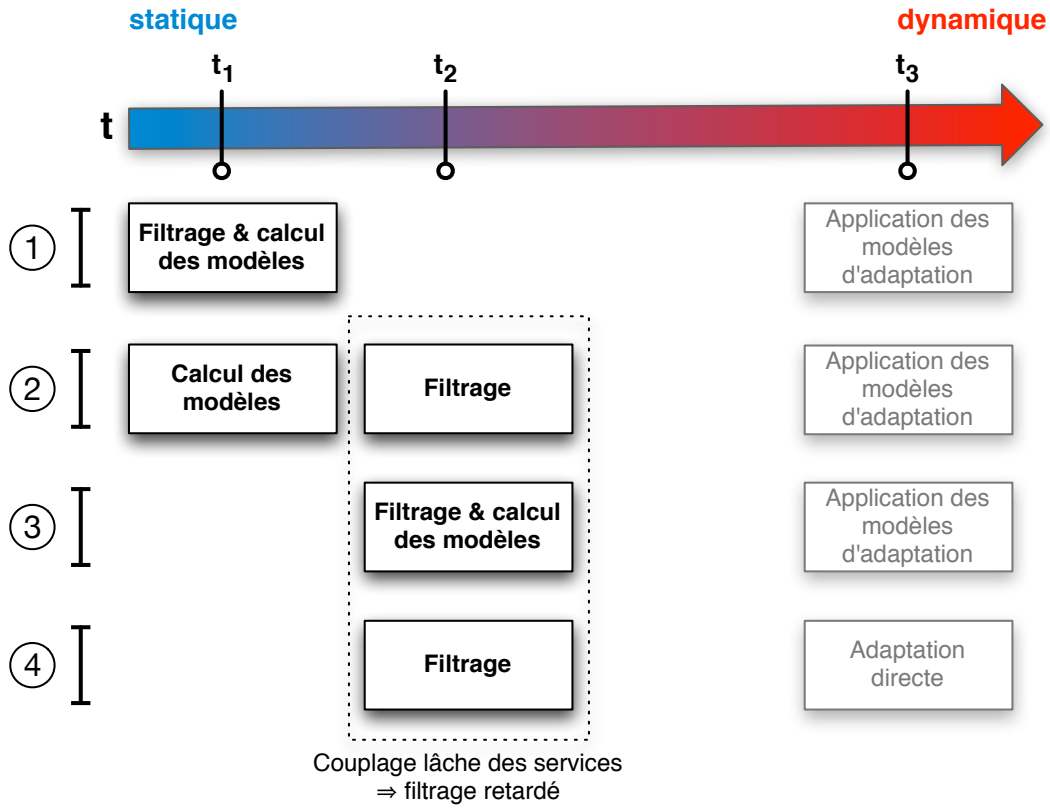

FIGURE 5.8 – Répartition des tâches de filtrage et de calcul des modèles d'adaptation de données.

où les offres de services ne sont pas encore connues lors de la définition du processus en *t1*. L'adjonction d'un cache pour conserver les adaptateurs entre chaque exécution d'un processus est avancé dans ces travaux pour optimiser leur génération, dans l'optique d'utiliser les même algorithmes mais cette fois pendant l'exécution des processus métiers. Cependant, ces améliorations restent difficilement applicables dans le contexte qui est le notre où les services et leurs types de données sont amenés à varier entre chaque exécution.

– La seconde option découple le calcul des modèles d'adaptation effectué en *t1* de l'étape de filtrage des services à proprement parler qui n'intervient que dans un second temps en *t2*, c'est-à-dire lors du chargement du processus métier. Cette approche, tout comme les deux suivantes, est plus appropriée dans un contexte SSOA particulièrement dynamique : le filtrage n'intervenant que lors du chargement du processus, il est en mesure de ne considérer que les services effectivement disponibles à cet instant. La complexité du calcul des modèles d'adaptation est effectivement reléguée au moment de la définition des offres de services et des processus de façon à ce qu'elle n'influe en rien sur l'exécution de ces derniers, tout comme dans la première approche. C'est par ailleurs l'approche qui est implicitement préconisée par la spécification SAWSDL et mise en œuvre dans nos précédents travaux sur une une architecture pour la découverte et l'orchestration de services Web sémantiques [Châtel, 2007a, Châtel, 2007b] : le calcul des modèles d'adaptation est possible avant même de filtrer les services dans la mesure

où il fait intervenir un modèle pivot de données auxquels doivent se référer tous les producteurs et consommateurs de services. Il s'agit d'ailleurs là de sa principale faiblesse car il nécessite une forte implication des différents protagonistes pour décrire à l'avance les transformations de données rattachées aux annotations *lifting* et *loweringSchemaMappings* des offres et demandes de services SAWSDL. Par ailleurs, ces transformations doivent logiquement être décrites pour *tous* les types de données considérés.

– La troisième alternative opte pour une co-localisation temporelle du filtrage des services et du calcul des modèles d'adaptation en *t2*, et ce pour un maximum de dynamicité. Contrairement à l'approche précédente, les modèles d'adaptation ne doivent être calculés que pour les sous-ensembles de services effectivement filtrés au chargement d'un processus, et non dans leur intégralité. Cependant, ce calcul étant initié lors du chargement du processus, il doit être effectuée de manière intégralement autonome et automatique (plus d'opérateurs humain pour le guider à cet instant) et ce dans un temps contraint : l'utilisation d'algorithmes complexes d'appariement n'est plus une option. Concrètement, il s'agit donc de mettre ici à l'œuvre des techniques d'adaptation de données plus simples et moins coûteuses en temps car le processus métier ne doit pas avoir à s'arrêter pour attendre que les modèles d'adaptation de données soient calculés. Des services qui auraient été appariés par l'utilisation d'algorithmes plus complexes peuvent ne plus l'être. Cependant, cet inconvénient est contre-balancé par la diversité de choix induite par le paradigme SSOA : de nombreux services fonctionnellement équivalents sont sensés entrer en concurrence au sein d'une infrastructure sous-jacente hautement dynamique.

– La quatrième et dernière option positionne, tout comme dans les deux précédentes approches, l'étape de filtrage des services (et donc de constitution des multiples ensembles de services candidats) en *t2*, lors du chargement du processus. Cette option présente cependant l'originalité de ne pas pré-calculer de modèles d'adaptation de donnée, mais se contente de faire une adaptation directe des données transmises entre processus et services lors de l'exécution. Dans ce cas de figure, seul un mécanisme d'adaptation *particulièrement basique* (et donc rapide) peut alors être implanté : on cherche en effet à minimiser l'impact très direct sur le temps d'exécution du processus de l'algorithme d'adaptation des données, ces dernières étant transformées "sur le fil".

Pour finir, tel qu'illustré dans la figure 5.5, à l'issue de cette étape de filtrage fonctionnel et non-fonctionnel des services, les informations techniques nécessaires à l'invocation ultérieure en liaison tardive de chaque service retenu sont obtenues (les offres de service retenues et les éventuels modèles d'adaptation de données), ainsi que celles indispensables à la mise en place de leur supervision (les contrats non-fonctionnels négociés entre le processus client et les services qui décrivent les multiples dimensions de QoS mesurables). Cette remarque étant par ailleurs valable quelque-soit l'option retenue parmi les trois dernières alternatives viables pour le calcul des modèles d'adaptation de données.

5.5 Supervision des services candidats

La supervision des services (on parle aussi de *"monitoring"*) est l'étape de notre approche par laquelle des informations non-fonctionnelles que l'on considère comme cruciales aux prises de décision

de liaison ultérieures sont recueillies et exposées. Il s'agit donc de répondre ici à deux questions diffuses concernant cette supervision : le *"comment ?"* puis le *"quand ?"*, et ce dans notre contexte SSOA à faible couplage.

5.5.1 Mise en œuvre de la supervision

Nous présentons ci-dessous trois alternatives concrètes pour la réalisation de la supervision au sein de la composition active de services :

1. *Une participation volontaire des fournisseurs de services*, où l'on dispose éventuellement d'un tiers de confiance. Dans ce cas de figure, ce sont alors les services eux-mêmes qui vont fournir en continu les valeurs courantes de chacune de leurs propriétés de QoS.

2. *Une supervision réalisée au niveau de l'intergiciel SSOA* et de son bus de services (dans le cadre de SemEUsE, il s'agirait de Petals [13]), en supposant effectivement que le processus métier client s'exécute sur le même bus que celui où les services candidats seront enregistrés. Les données ainsi recueillies pourraient aussi être capitalisées et partagées à plusieurs niveaux : par exemple entre les processus métiers qui s'exécutent sur un même bus, entre plusieurs clients ayant chacun plusieurs processus métiers à exécuter, entre de très nombreux processus métiers dans le cas d'un bus s'exécutant sur une grappe de machine de type "cloud", etc.

3. *Une supervision effectuée par le processus métier lui-même*, qui accumule des données lors de chaque appel de services, en suivant donc une approche d'apprentissage en ligne. Il s'agit donc moins dans cette option de QoS *courante* que de valeurs résultants d'un calcul effectué à partir des valeurs obtenues depuis le début de l'exécution du processus.

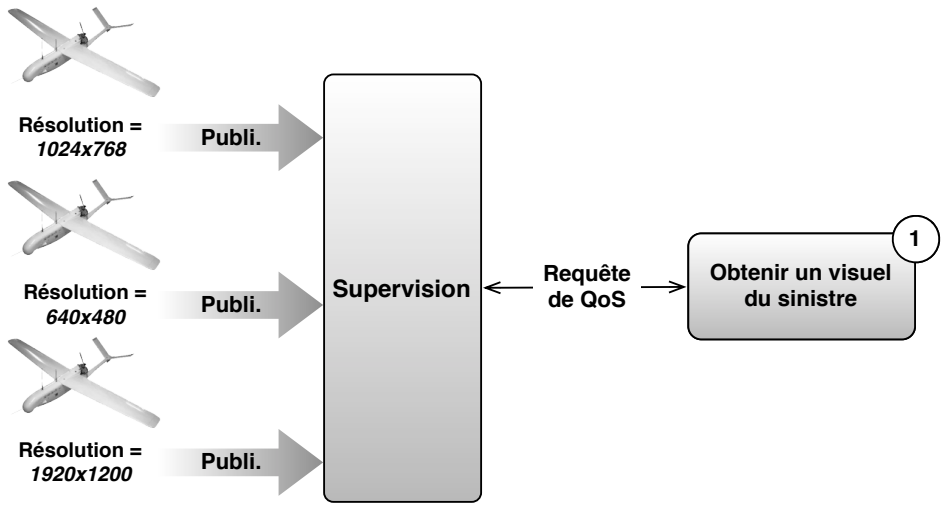

FIGURE 5.9 – Participation volontaire des drones à la supervision.

C'est la première des trois options qui a été retenue dans notre approche, elle est illustrée schématiquement par la figure 5.9 où les drones rendent volontairement disponibles leurs valeurs courantes

13. http ://petals.ow2.org/

de QoS en terme de résolution d'image, via un canevas de supervision, à partir duquel les valeurs courantes de QoS sont ensuite récupérées pour effectuer une décision de liaison de service. Il faut cependant noter qu'elles ne sont pas antinomiques et pourraient conjointement faire partie d'une approche plus globale, bien que probablement redondante, de la supervision. Cette participation volontaire est viable car nous partons du constat qu'un service donné aura avantage à publier sa Qualité de Service courante si cette dernière est bien meilleure que celle portée comme extremum dans son contrat non-fonctionnel précédemment annoncé. Ce qui sera le cas si le fournisseur a pris de grandes marges dans le contrat afin de ne pas avoir à payer de pénalités en cas de violation des garanties, mais qu'il fourni habituellement en pratique une bien meilleure QoS. Dans ce contexte, concurrence entre services aidant, ses semblables suivront si les consommateurs de services plébiscitent les fournisseurs qui fournissent ces valeurs courantes.

Bien que cette option puisse malgré-tout sembler avant-gardiste compte tenu de l'état actuel du déploiement de ce type de technologies dans le monde de l'entreprise, elle risque de devenir inévitable, à plus ou moins courte échéance, si les besoins actuels et bien réels en termes de *certification* des activités liées aux technologies non-fonctionnelles de services se concrétisent au niveau industriel. De fait, quand les fournisseurs de services chercherons à obtenir des certifications de type *ISO-900X*, fort est à parier que la possibilité de vérifier la qualité du service rendu (ou perçu) fera partie intégrante de ces certifications ; la capacité à *s'auto-évaluer* en est le plus souvent une caractéristique essentielle. Ce modèle économique, fondé sur la crédibilité des fournisseurs de services, ne peut alors que renforcer la pertinence de la participation volontaire de ces derniers au processus de supervision.

Ainsi, une hypothèse centrale à notre approche est que les contrats non-fonctionnels, "signés" par les fournisseurs de services suite aux négociations effectuées lors du précédent filtrage, expriment les propriétés de QoS qu'il est possible de superviser tout au long de l'existence des services. Cependant, si la supervision des services est effectivement un moyen particulièrement adapté à l'obtention des valeurs pertinentes de QoS pour la prise de décision de liaison, une alternative sensiblement dégradée consisterait à toujours considérer lors de la décision la "pire" valeur possible de QoS selon les bornes définies dans le contrat statique. Cette approche pessimiste, qui n'a pas été retenue pour nos travaux, pourrait néanmoins y être utilisée comme mécanisme de secours en cas de défaillance technique du canevas de supervision.

5.5.2 Un ensemble de contraintes spécifiques

L'approche générale pour la supervision des services étant fixée, il subsiste un ensemble de contraintes, propre à la prise de décision dynamique, qui s'y applique. Ces contraintes portent sur :

– *la cohérence des données.* En effet, la comparaison entre plusieurs services étant le plus souvent effectuée sur la base des valeurs courantes de multiples propriétés de QoS, les données relevées doivent exhiber un certain degré de cohérence sur le plan temporel. Cette cohérence doit se retrouver aussi bien entre les différentes propriétés d'un même service (cohérence *interne*), qu'entre les services considérés (cohérence *externe*). Par exemple, selon notre cas d'utilisation, on doit être en mesure d'obtenir à tout moment la valeur courante de la bande passante B et de la résolution d'image R d'un drone de surveillance donné. Lors de la comparaison entre drones dans l'optique d'une décision de liaison, il s'agit alors d'une part de ne pas mettre sur un pied d'égalité deux drones D_1 et D_2 dont le degré d'obsolescence des valeurs obtenues ne serait pas le même (par exemple si D_1 offre une meilleure valeur B_1 sur la bande passante

que D_2, mais que B_2, bien qu'inférieure, est beaucoup plus récente donc pertinente pour la décision) ; et, d'autre part, de s'assurer de la proximité temporelle interne des valeurs sur B et R au sein de D_1, D_2 et tous les autres drones ;

– *la flexibilité de l'accès aux données.* Pour une même propriété de QoS, chaque fournisseur de services peut utiliser son propre "vocabulaire", matérialisé dans le contexte SSOA par des concepts ontologiques (voir des ontologies) distincts, ou ses propres unités de mesure. Nous avançons que c'est au canevas de supervision de gérer ces discordances, tout en les masquant à ses clients (en l'occurrence le processus de composition active de services). Toujours selon notre cas d'utilisation, de telles différences subtiles pourraient apparaître entre les offres de services issues des pompiers et des gendarmes pour un même type de matériel : par exemple, des camions citernes dont la vitesse est caractérisée par le concept ontologique *Vitesse* et mesurée en Km/h d'un côté, et le concept *Speed* en mph [14] de l'autre.

– *la performance dans l'accès aux données.* Le processus décisionnel, client des valeurs fournies par le canevas de supervision, ne doit pas pâtir d'éventuels délais dans l'accès au valeurs de QoS. Ces délais proviennent essentiellement des contraintes techniques liées à la transmission des valeurs numériques depuis les services ainsi qu'à leur éventuelle conversion. Or, dans les cas usuels, les délais de transmission vers les clients sont proportionnellement importants (par exemple de l'ordre de plusieurs milli-secondes) par rapport à celui de la décision effectuée en local.

Le canevas de supervision M4ABP (*"Monitoring for Adaptive Business Process"*), mis au point par Le Duc *et al.* [Le Duc et al., 2009] dans le cadre du projet SemEUsE et de travaux connexes à cette thèse apporte de nombreux éléments de réponse à ces contraintes en mettant en œuvre différentes techniques telles la mise en cache des valeurs de QoS obtenues des services pour en augmenter la performance d'accès, ainsi que leur marquage temporel ("timestamp") pour assurer une certaine cohérence temporelle des données. Il repose par ailleurs sur l'obtention des valeurs courantes de QoS que chaque service s'engage à fournir volontairement au travers d'une interface dédiée. C'est ce canevas qui sera intégré à l'implantation de la composition active des services.

5.5.3 Intégration au processus de composition active

Si la réponse aux contraintes précédemment définies est en partie fournie par l'implantation du canevas de supervision, son utilisation habile et ciblée constitue un second vecteur d'optimisation en termes de performance globale du processus de composition de services. En effet, du point de vue externe de ce dernier, les interactions avec le canevas de supervision vont être séparées en deux étapes distinctes :

– Tout d'abord *l'initialisation de la supervision*, qui entraîne son déclenchement. Elle doit être effectuée le plus tôt possible (en amont de l'exécution du processus métier) afin de ne pas en influencer le bon déroulement.

– Vient ensuite *le recueil des informations de supervision* en tant que telles (c'est-à-dire celui des valeurs courantes de QoS des services supervisés) qui sera effectué dans cette approche au moment ultime de la sélection de service avant invocation, de manière à obtenir les valeurs de QoS les plus fraîches possible (cf. section 5.6).

14. "mile per hour", aussi noté mi/h, unité de vitesse issue du monde anglo-saxon

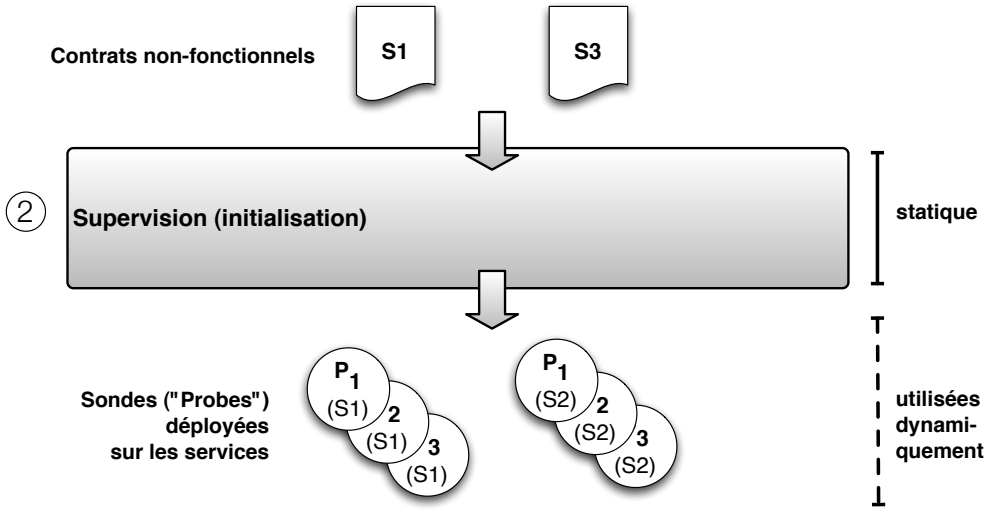

FIGURE 5.10 – Initialisation de la supervision des services ayant été filtrés.

Nous proposons donc, comme optimisation dans l'usage du canevas, d'effectuer son initialisation de manière précoce, dans la foulée de l'étape de filtrage des services, ce qui permet aussi de la restreindre aux seuls services exhibant les propriétés non-fonctionnelles contractuelles nécessaires. L'initialisation et le déclenchement de la supervision seront donc, tout comme le filtrage, effectués de manière statique au chargement du processus (en *t2* sur la figure 5.8), avant l'exécution de sa partie métier.

Par la suite, le coût du recueil des informations non-fonctionnelles est en grande partie minimisé par ce déclenchement préalable en *t2* du canevas de supervision. En effet, à sa suite, des sondes ont été déployées sur la base des informations techniques contenues dans les contrats non-fonctionnels, tel qu'illustré sur la figure 5.10. Le rôle de ces sondes est de s'abonner aux interfaces de supervisions locales à chaque service de manière à faire remonter, *de manière asynchrone* à l'exécution du processus métier, les valeurs courantes de Qualité de Service pour consultation ultérieure. Cette asynchronicité permet de s'affranchir, lors de la prise de décision de liaison dynamique à l'exécution du processus, des délais relatifs à la demande, puis l'obtention *via* le réseau des valeurs courantes de QoS des services.

5.6 Liaison tardive d'un service

L'étape de liaison tardive d'un service, que l'on peut décomposer en une sélection tardive d'un service spécifique suivie de son invocation, est effectuée juste-à-temps, lors de l'évaluation d'une source d'appel de services pour un processus donné. Ce faisant, on passe d'une démarche statique mise en œuvre pour les étapes préliminaires de filtrage et d'initialisation de la supervision, à une approche fortement dynamique. On dispose alors, à cet instant, des valeurs de QoS les plus "fraîches" possibles, obtenues *via* le canevas de supervision préalablement déployé : après la première étape de filtrage qui a permis l'élaboration d'ensembles distincts de services candidats pour chaque source d'appel, on peut donc utiliser une nouvelle fois des informations non-fonctionnelles, mais cette fois

plus pertinentes, pour diriger la décision finale de liaison entre le processus et un service. On est aussi en mesure de prendre en compte la disponibilité effective des services : en effet, un service candidat préalablement filtré pourrait ne plus être marqué comme disponible à cet instant précis. Cette étape dynamique de sélection est alors garante, dans le prolongement du filtrage de services, de l'agilité de notre approche pour la composition de services : l'exécution d'un processus métier est ainsi capable de prendre en compte, au mieux, l'état actuel de son contexte d'exécution.

Cependant, si le cadre technique ainsi posé répond effectivement de l'agilité de l'approche, il n'est pas suffisant, à lui seul, pour effectuer une décision de liaison "performante" : un mécanisme d'intégration intelligente des propriétés non-fonctionnelles doit être défini. Ce mécanisme a la charge de diriger la décision, de manière à maximiser l'*utilité* de chaque liaison processus/service, et, *in fine*, la performance globale du processus. Pour ce faire, les besoins non-fonctionnels usuels, exprimés sous forme de contrats, sont complétés dans notre approche par des préférences utilisateur qui permettent au programmeur d'un processus métier d'exprimer ses politiques de décision dans la liaison et l'utilisation des services physiques en fonction des valeurs courantes de Qualité de Service. Ces dernières sont établies, statiquement, avant le chargement de leur processus parent, et les valeurs courantes de QoS leur sont appliquées lors de son exécution. Cette approche, qui permet une *composition utile de services* en s'imbriquant au cœur de la liaison tardive des services, les préférences non-fonctionnelles qui en dépendent, et le principe de raisonnement sur ces préférences, sont présentés en détails au chapitre 6.

Les préférences sont aussi tout à fait adéquates utiles pour gérer certains besoins pouvant se révéler contradictoires en termes de QoS (par exemple *prix* vs. *qualité*), car elles permettent d'établir la préférence relative entre les propriétés non-fonctionnelles supervisées : ainsi, pour chaque ensemble de services candidats, un ordre total sur les services peut être obtenu, et les décisions finales de liaison peuvent alors être effectuées automatiquement, sans nécessiter d'arbitrage à l'exécution par un opérateur humain.

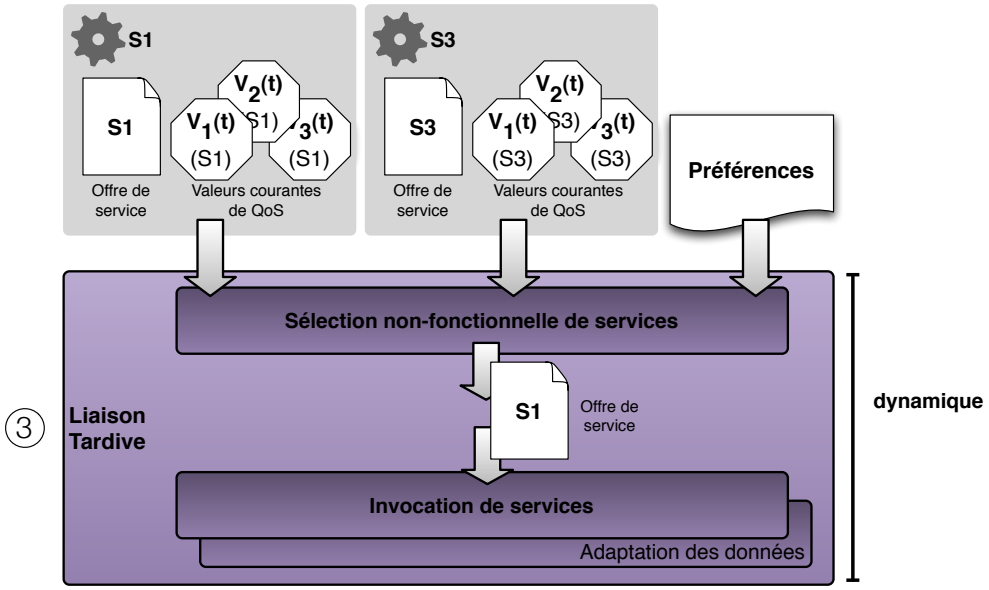

FIGURE 5.11 – Sélection et invocation d'un service en liaison tardive.

Une fois la sélection d'un service effectuée, à partir de sa QoS courante et des préférences, son invocation constitue l'ultime phase de la liaison tardive de services. Elle consiste à envoyer un message d'appel de service au travers du réseau et en exploiter la ou les valeur(s) de retour en appliquant les modèles d'adaptation de données qui ont été calculés au préalable. Ces différents paliers sont dépeints dans la figure 5.11. Il est par ailleurs à noter que si plusieurs services sont sélectionnés au lieu d'un, c'est qu'il sont alors, à cet instant précis, strictement équivalents sur les plans fonctionnels et non-fonctionnels : la probabilité pour que cette configuration apparaisse est très faible, mais, le cas échéant, un de ces services sera choisi au hasard pour invocation. Cette démarche permet de minimiser les risques de famines des services ("starvation") bien connus en gestion de la concurrence [Cleaveland et Smolka, 1996].

Par ailleurs, le précepte d'agilité ici présenté s'applique aussi à la définition des préférences utilisateur établies pour la composition de services. En effet, s'il est nécessaire dans ce contexte que les préférences soient établies statiquement, cela ne signifie pas pour autant qu'elles doivent l'être conjointement à la définition de leur processus parent. De fait, la pertinence des préférences est d'autant plus importante que ces dernières ont été adaptées au contexte de déploiement de leur processus ; ce contexte restant la plupart du temps indéterminé au moment de sa définition.

L'influence du contexte de déploiement et l'intérêt d'une définition tardive (et par là-même particulièrement dynamique) des préférences peuvent être facilement illustrés. En prenant pour exemple notre cas d'utilisation en lutte anti-incendie (cf. chapitre 4), une préférence simple, nommée P, concernant les camions de pompier considérés à un instant précis serait de chercher à maximiser simultanément leur *contenance en eau* courante, appelée E, et leur rayon d'action possible, tributaire de leur *réserve en carburant* courante, appelée C. Mais si l'on cherche, dans l'absolu, à maximiser E et C, la préférence relative entre ces deux caractéristiques va dépendre du contexte de déploiement : dans le cas d'un incendie de grande ampleur en milieu naturel on va vouloir augmenter la couverture du dispositif anti-incendie en favorisant les camions disposant d'un grand rayon d'action au détriment de ceux disposant d'une plus grande réserve en eau. Cette relativité entre E et C sera alors exprimée dans la préférence rattachée à la source d'appel aux services "camions de pompier" de ce micro-processus de gestion d'incendie.

Par conséquent, en se reportant à l'échelle définie par la figure 5.7, les préférences devront être spécifiées et rattachées à leurs processus respectifs à un instant t_{prefs} compris entre la définition de ces derniers en t_1 et leur chargement en t_2 ; où plus précisément : $t_{prefs} \in [t_1, t_2[$.

5.7 Conclusion

Dans ce chapitre nous avons présenté, le plus souvent de manière abstraite, les fondations de notre contribution pour la composition active de services. Le but de cette approche est cependant de répondre à un besoin bien concret, celui d'améliorer les capacités d'adaptation des systèmes informatiques complexes. En tant qu'incarnation de ces systèmes complexes, les applications réparties reposant sur une architecture SOA et plus particulièrement SSOA sont, de fait, de plus en plus déployées "sur le terrain".

Dans ce contexte, la *dynamicité* et la *flexibilité* d'une application est évaluée à l'aune de sa capacité à réagir promptement aux constantes évolutions de son contexte d'exécution, ce qui est grandement facilité par la liaison tardive de services, sur la base des valeurs courantes de QoS, que

nous avons mis en place. De plus, une implantation de cette liaison tardive a été effectuée et sera présentée au chapitre 9.

La *performance* de l'application est quant à elle évaluée sur sa capacité à choisir et à utiliser les services les plus "utiles" à son exécution, ce que nous aborderons au chapitre 6.

Chapitre 6

Composition utile de services

Sommaire

6.1 Introduction

Nous avons précédemment détaillé les étapes permettant d'implanter, dans le cadre d'une architecture (S)SOA, un niveau élevé d'agilité dans la composition de services, tout en minimisant son impact sur les performances des applications en tirant profit d'une séparation entre les moments statiques et dynamiques jalonnant le cycle de vie d'un processus métier. On cherche alors maintenant, non plus à préserver, mais à améliorer les performances de ces systèmes complexes tout au long de leur exécution. Cette amélioration passe ici par la prise en compte des préférences non-fonctionnelles des utilisateurs lors de l'ultime étape de composition : la sélection et l'invocation tardive d'un service. Cette prise en compte va permettre de maximiser l'*utilité* de la liaison entre un processus et un service ; et, ce faisant, d'améliorer la performance du processus dans son ensemble par une multitude d'optimisations locales à chaque site d'appel de service.

Ce chapitre a donc pour vocation de poser les bases conceptuelles d'une composition *utile* de services qui passe notamment par l'introduction d'un nouveau formalisme de préférences, celui des

LCP-nets ("Linguistic CP-nets"), qui va permettre au programmeur d'un processus métier d'exprimer, *a priori*, ses politiques de décision dans la liaison et l'utilisation des services physiques en fonction des valeurs de Qualité de Service effectives (ou "courantes") de ces derniers, obtenues lors de l'exécution du processus.

Cependant, si les LCP-nets s'avèrent particulièrement adaptés au monde des SSOA, leur formalisme n'est, par nature, lié à aucun domaine de décision multi-critères en particulier : c'est en ce sens que l'on peut qualifier le formalisme LCP-net de générique. Partageant la même philosophie, le mécanisme par lequel les préférences ainsi exprimées sont transformées et évaluées, peut lui aussi être mis à l'œuvre en dehors de ce contexte spécifique.

En complément de cette *généricité*, nous illustrerons aussi dans les sections suivantes la *simplicité* d'utilisation du formalisme, telle que définie préalablement comme un des objectifs principaux de cette thèse. En effet, en l'absence dans notre contexte d'experts de l'élicitation de préférences, ce formalisme présentera la double caractéristique d'être d'une part qualitatif, par l'utilisation de concepts linguistiques, afin de gérer au mieux les imprécisions et incertitudes inhérentes à la modélisation de ce type de préférences, et d'autre part la capacité à exprimer les éventuelles mais potentielles contradictions entre plusieurs propriétés non-fonctionnelles offertes par un même service.

Ce chapitre est organisé comme suit : dans un premier temps on introduit en 6.2 la problématique d'une modélisation qualitative des préférences non-fonctionnelles ainsi que la réponse qui y est apportée par le nouveau formalisme des LCP-nets que nous proposons dans la section 6.3. S'en suit en 6.4 une présentation détaillée des mécanismes que nous mettons actuellement en œuvre, depuis l'élicitation de préférences jusqu'à la sélection de services. Nous nous efforcerons par ailleurs de mettre la présente approche en perspective tout au long de ce chapitre, en abordant l'intérêt d'un futur traitement entièrement linguistique des préférences, ainsi que son impact sur la démarche actuelle dans la section 6.5.

6.2 Problématique d'une modélisation qualitative des préférences non-fonctionnelles

Nous avons choisi de fonder nos travaux de modélisation sur la famille *CP-nets de langages de représentation compacte de préférences qui ont été préalablement détaillés dans l'état de l'art de ce manuscrit (cf. section 3.2.2). Ce choix a été effectué car ces modèles offrent des avantages notables dans notre contexte applicatif, tels que leur facilité d'utilisation pour les modélisateurs (non-experts) de préférences, leur coût computationnel relativement faible, leur structuration robuste et leur potentiel manifeste en termes d'extensions pour le support de propriétés additionnelles qui pourraient s'avérer capitales vis-à-vis de notre contexte. D'autres formalismes ont été considérés, tels les réseaux GAI (cf. section 3.2.3), mais ces approches, de par l'effort important sur l'étape d'élicitation ainsi que le coût en termes de raisonnement qu'elles impliquent, ne sont pas adaptées à notre contexte. *A contrario*, les GAI-nets sont par contre particulièrement adaptées pour effectuer une discrimination sur une quantité très importante d'alternatives, ce qui n'est pas le cas ici.

Cependant, les *CP-nets présentent deux limitations importantes pour l'approche dynamique de gestion de la Qualité de Service que nous souhaitons mettre en œuvre.

Tout d'abord, la plupart des dimensions de QoS que nous sommes amenées à manipuler sont définies sur des domaines continus de valeurs numériques, mais la famille des *CP-nets dans son

ensemble, que ce soient les CP-nets, UCP-nets ou TPC-nets, ne sait traiter que le cas discret des domaines finis de variables. *Nous proposons alors d'effectuer une discrétisation systématique (un partitionnement flou) des domaines continus, par l'utilisation de termes linguistiques flous [Zadeh, 1975], au lieu d'ensembles précis* ("crisp sets", au sens de la théorie des ensembles). Cette approche évite d'ériger des barrières très strictes et artificielles entre les valeurs des domaines en permettant un passage lissé d'une classe de valeurs à une autre, et en permettant d'indiquer les imprécisions dans la détermination de ces classes. Par ailleurs, cette démarche évite la perte d'information liée à la discrétisation puisqu'il s'agit de conserver un intervalle continu pour les calculs bas niveau d'inférence, mais de faire apparaître au plus haut niveau (de l'utilisateur final) les termes linguistiques associés aux classes.

C'est pourquoi, dans un contexte où les utilisateurs doivent exprimer leurs préférences entre des valeurs de domaines continues, la nature qualitative des ensembles flous, disposant de transitions progressives, s'est révélée capable de mieux capturer leur intention. Par conséquent, cette approche devrait aussi permettre une sélection plus *utile* entre les services lorsqu'ils disposent à un instant donné de propriétés de QoS aux valeurs relativement proches, en évitant de mettre de grandes différences artificielles de préférence entre ces services qui offrent des caractéristiques non-fonctionnelles qu'on pourrait considérer, toutes proportions gardées, comme étant globalement similaires.

Un autre écueil propre aux *CP-nets est que l'obtention de valeurs *précises* d'utilité de la part d'utilisateurs non-spécialistes des domaines considérés est vouée à l'échec, ou pire, induit le recueil d'informations involontairement erronées. En effet, pour ces utilisateur, donner un nombre particulièrement précis pour exprimer une perception (ou une préférence dans notre cas de figure) n'est pas toujours possible [Zadeh, 1975, Degani et Bortolan, 1988, Herrera et Martínez, 2000, Truck et Akdag, 2006]. Face à cette problématique, les *CP-nets fournissent seulement deux alternatives.

- D'un côté, le modèle CP-net de base permet l'expression de préférences rudimentaires, bien qu'intuitives, *via* l'établissement d'une simple relation d'ordre directement entre variables discrètes de Qualité de Service, évacuant ainsi la notion même d'une quantification de l'utilité des services. Il souffre cependant de faibles performances lors de la comparaison de deux affectations.

- De l'autre, le modèle UCP-net permet d'effectuer efficacement ces comparaison d'affectations, mais contraint pour cela l'utilisateur à définir préalablement des valeurs numériques précises d'utilité pour les services. Valeurs dont l'obtention, comme nous l'avons précédemment établi, est particulièrement problématique par rapport à une simple relation d'ordre.

Prises séparément, aucune de ces deux approches n'est satisfaisante dans notre contexte. De plus, elles ne sont adaptées qu'au cas discret, car la définition d'une relation d'ordre sur des ensembles continus de QoS ne peut se faire par de simples tables ; ce qui conforte notre première proposition de partitionnement flou des domaines de définition des valeurs QoS. *Une seconde proposition de cette thèse est alors de conserver la notion d'utilité tout en étendant l'approche qualitative à l'expression de ses valeurs*, et donc de ne pas cantonner son application aux dimensions de QoS.

Par ailleurs, nous faisons reposer la comparaison entre les affectations de préférences sur celle de leurs valeurs globales d'utilité, analogues à celles promues par les UCP-nets, mais qui seront dorénavant calculées par la logique floue et des outils d'agrégation adaptés.

6.3 Linguistic CP-nets (LCP-nets)

Nous avançons ici une nouvelle déclinaison des CP-nets, appelée LCP-nets ("Linguistic Conditional Preference networks") [Châtel et al., 2008, Châtel et al., 2010b], dans le but de bénéficier d'un mariage entre :

- *l'approche linguistique* (cf. section 3.1), mise en œuvre au travers d'une modélisation floue des concepts (cf. section 3.1.1) ;
- *l'approche graphique*, promue par les *CP-nets (cf. section 3.2.2) ;
- *certaines propriétés propres aux UCP-nets et aux TCP-nets*, telles la possibilité d'évaluer l'utilité d'une affectation ou celle d'exprimer des compromis.

En réponse à la problématique d'une modélisation qualitative des préférences non-fonctionnelles, les LCP-net réalisent l'application d'une approche linguistique floue, de manière à obtenir un modèle qualitatif unifié pour l'expression de préférences. Conceptuellement, ils permettent ainsi de définir, par un ensemble fini de règles exprimées dans leurs tables, une relation d'ordre sur des ensembles continus.

Comparativement aux précédents modèles *CP-nets [Boutilier et al., 2004, Boutilier et al., 2001, Brafman et Domshlak, 2002], ce nouveau formalisme permet la modélisation sous forme graphique de déclarations plus qualitatives que l'on pourrait retranscrire par exemple par *"Je préfère une valeur* approchant V_1 *à* exactement V_2 *pour la propriété* X*, si une autre propriété* Y *vaut* approximativement V_Y *et* Z *vaut* un peu plus que V_Z*"*. De plus, ces déclarations, que l'on peut assimiler à des règles floues élaborées, sont interprétées dans un contexte où la préférence sur X doit prendre en compte toute autre déclaration qui s'applique à la valeur de Y.

Comparativement aux modèles linguistiques existants pour la modélisation de préférences [Herrera et al., 2009, Wu et Mendel, 2007, Tong et Bonissone, 1980, Yager, 1981, Buckley, 1984, Shendrik et Tamm, 1986], les LCP-nets sont conditionnels (la préférence entre les alternatives peut maintenant dépendre d'un facteur externe) et graphiques, ils permettent donc d'exprimer des préférences plus complexes et moins ambiguës que sous la forme de phrases.

6.3.1 Un ensemble de contraintes à prendre en compte

Les contraintes et propriétés particulières du contexte SSOA ont dû être intégrées lors de la conception des LCP-nets. On note tout particulièrement que :

- *les préférences doivent être simples à définir* puisque, dans les contraintes qui s'imposent à nous, les programmeurs de processus métiers ne peuvent pas s'appuyer sur des experts de modélisation de préférence, ni ne disposent de beaucoup de ressources à allouer pour leur élicitation. Ce qui implique qu'un certain degré d'imprécision doit être toléré dans les modèles de préférences ;

- *les problèmes à traiter ne font habituellement usage que de peu de variables* (de l'ordre de 10 variables).

- dans ce contexte, *le temps de calcul pour la prise de décision fondée sur les préférences est une contrainte faible*, dans la mesure où celui-ci reste reste relativement faible par rapport au délai de l'invocation subséquente de service *via* le réseau.

Elles nous amènent à développer les caractéristiques suivantes de notre formalisme de préférence : il est *graphique* pour faciliter leur définition sans déraisonnablement compromettre leur calcul, puisque les modèles LCP-nets sont bien plus commodes à établir que de devoir écrire plusieurs ensembles de règles floues potentiellement interdépendantes, et il est *qualitatif* de manière à faire face à l'imprécision de l'utilisateur ou des valeurs remontées par les sondes.

Pour répondre à cette dernière caractéristique, des variables linguistiques [Zadeh, 1975], ont été incorporées dans les LCP-nets, la sémantique de chacune d'entre elles étant donné par un ensemble flou. Par conséquent, les modeleurs de préférence peuvent manipuler facilement des termes linguistiques pré-existant durant l'élicitation. Par ailleurs, comme dans tout modèle graphique, les LCP-nets possèdent des nœuds qui correspondent aux variables du problème et dont les domaines continus sont discrétisés sous forme d'ensembles de termes linguistiques. Dans le contexte des SSOA, ces variables correspondent alors aux propriétés non-fonctionnelles des services.

En résumé, les LCP-nets permettent aux utilisateurs d'exprimer des compromis entre les variables par l'utilisation d'*i-arcs* ou *ci-arcs* hérités des TCP-nets, possèdent des tables CPTs similaires à celles des UCP-nets, mais expriment les utilités avec des termes linguistiques plutôt qu'avec des valeurs numériques. Il est ainsi possible avec les LCP-nets :

- d'éliciter les affectations préférées pour un domaine spécifique de QoS (ou plusieurs domaines interdépendants), par l'utilisation de CPTs similaires à celles des UCP-nets [Boutilier et al., 2001] ;
- de révéler l'importance relative des propriétés non-fonctionnelles, par l'utilisation d'arcs hérités des CP-nets ;
- d'indiquer les compromis entre propriétés non-fonctionnelles, par l'utilisation d'i-arcs et ci-arcs hérités des TCP-nets [Brafman et Domshlak, 2002].

6.3.2 Cas d'utilisation et LCP-nets

Le cas d'utilisation en lutte anti-incendie précédemment détaillé (cf. chapitre 4), fait état de préférences distinctes sur la sélection de drones de surveillance, de personnels au sol et de camions citerne de pompiers. Ces préférences, préalablement définies de manière textuelle (par exemple sur les camions, "on souhaite obtenir les meilleurs valeurs de vitesse et de réserve de carburant"), vont être maintenant modélisées sous forme de LCP-nets, de la manière la plus simple qui soit. Les domaines de QoS et d'utilité considérés ont subi au préalable un partitionnement flou et les termes linguistiques utilisés dans les tables de préférences (B_L, R_H, *low*, *very high*, etc.) obtenus.

Les propriétés graphiques et qualitatives du formalisme, mentionnées dans la section précédente, sont ainsi tout d'abord illustrées dans la figure 6.1, où la préférence d'un utilisateur pour le choix d'un drone d'imagerie aérienne est explicitée. Le but global de l'utilisateur, influencé par les pratiques courantes en gestion de crise environnementale, est d'obtenir les images le plus rapidement possible, de manière à pouvoir prendre une décision d'action au plus vite : en effet, des vies sont probablement en jeu. L'obtention de ce but sera alors notamment favorisée par une préférence d'optimisation de la bande passante au détriment du niveau de sécurité offert pour le transfert des données. Cet objectif est donc traduit en termes de préférences établies en fonction des trois propriétés de Qualité de Service supervisées sur les services : la sécurité (S), la bande-passante (B) et la résolution de l'image transmise (R). A la lecture des préférences, on peut ainsi constater que l'utilisateur va toujours

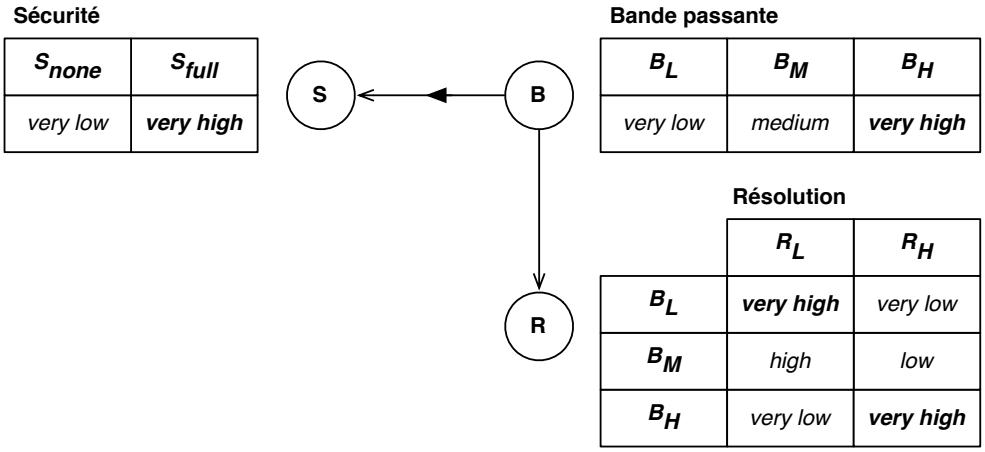

FIGURE 6.1 – Préférences sur la QoS d'un drone d'imagerie aérienne.

préférer la bande passante à la sécurité, et si la bande passante est faible, il va favoriser des images de faible résolution de manière à les obtenir en minimisant la latence.

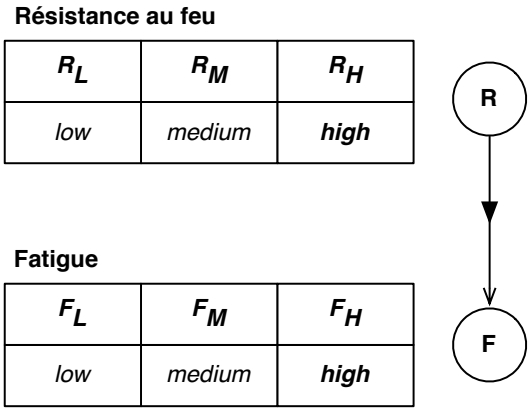

FIGURE 6.2 – Préférences sur la QoS d'un personnel au sol.

De la même manière, la retranscription compacte sous forme de LCP-net des préférences en termes de personnel au sol (qu'il s'agisse de pompiers ou de gendarmes) et illustrée par la figure 6.2. Les deux propriétés de QoS considérées sont la résistance au feu (R) et la fatigue physique (F). La préférence inconditionnelle relative de la résistance au feu par rapport à la fatigue physique est matérialisée par un i-arc tendu du nœud R au nœud F (représenté graphiquement par un arc avec un triangle noir en son milieu).

Pour finir, le LCP-net correspondant à la description textuelle des préférences sur un camion citerne d'intervention anti-incendies est représenté dans la figure 6.3. Contrairement aux deux modèles précédents, il apporte cependant une précision supplémentaire en effectuant une généralisation de la préférence exprimée. La description textuelle faisait en effet état d'une préférence inconditionnelle de la vitesse de déplacement (V) et de la réserve en carburant (C) par rapport à la contenance en eau

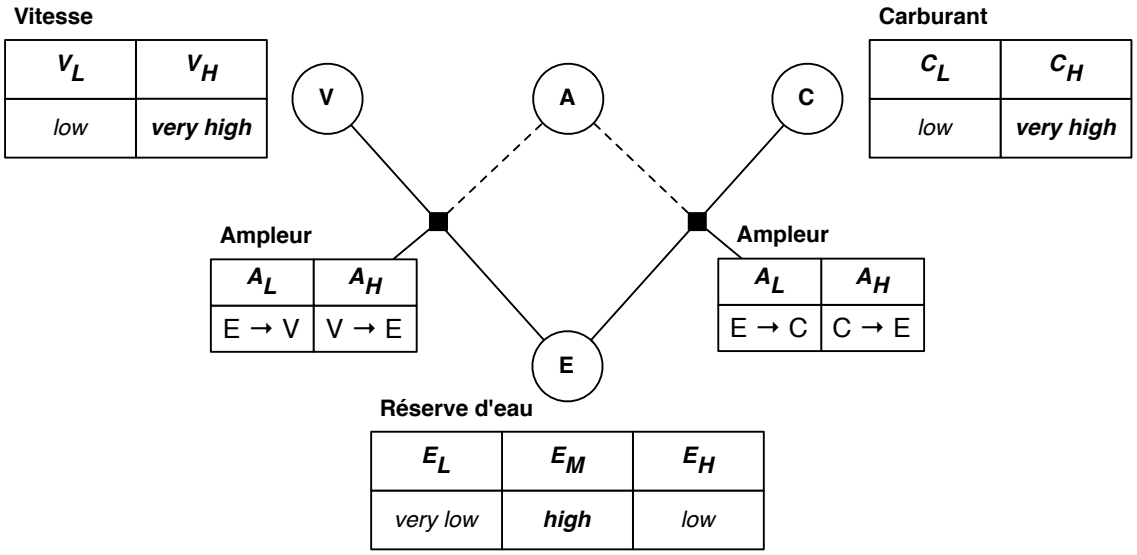

FIGURE 6.3 – Préférences sur la QoS d'un camion citerne d'intervention anti-incendies.

(E), cette préférence étant dictée par l'ampleur importante de la crise considérée. Dans ce LCP-net, on intègre la notion d'ampleur sous forme d'un nœud A, au même titre qu'une dimension classique de QoS, or il s'agit ici d'une *donnée contextuelle* obtenue hors du champs des services. Cet exemple ne fait alors qu'appuyer sur le caractère particulièrement flexible du formalisme LCP-net qui n'est aucunement limité à la seule intégration de dimensions de QoS dans un contexte SOA. Ce nœud A est lié à un ci-arc tendu entre V et E et un autre entre C et E, les tables qui y sont rattachées indiquent le sens effectif des relations en fonction de la valeur de A (A_L indique une faible ampleur, et A_H une grande ampleur). Ces deux préférences conditionnelles vont donc dépendre de l'ampleur du sinistre :

- s'il s'agit bien d'un sinistre de grande ampleur, les deux ci-arcs sont équivalents des i-arcs accordant une prédominance V et C sur E (on cherche à favoriser l'agilité et l'autonomie des camions) ;
- dans le cas contraire, c'est la capacité d'extinction qui sera mise en avant via le nœud E prédominant sur V et C.

6.3.3 Sérialisation XML

Enfin, tel que nous le verrons au chapitre 9, une implantation du langage LCP-net a été réalisée grâce au canevas EMF ("Eclipse Modeling Framework"). Le choix d'une sérialisation des modèles graphiques sous forme de fichier XML, facilitée par EMF, s'est alors imposé comme le prolongement naturel des formats couramment rencontrés dans les technologies SSOA. Ainsi, une fois les outils graphiques disponibles utilisés pour spécifier le LCP-net présenté à la figure 6.1, le fichier suivant au format XML est obtenu par sérialisation (cf. code 6.1). Il présente de manière intelligible l'ensemble des informations nécessaires à l'interprétation du modèle de préférences $ImagingServicePreference$, de la spécification des partitionnements flous des domaines aux tables de préférences conditionnelles.

```
<?xml version="1.0" encoding="UTF-8"?>
<LCPnet:LCPnet xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:LCPnet="http://www.thalesgroup.com/
     preferenceFramework/models/lcpnet" name="ImagingServicePreference">
  <nodes xsi:type="LCPnet:LNode" name="security" inArcs="//@arcs.0" valueDomain="//@valueDomains.0">
    <domain xsi:type="LCPnet:LNodeValue" name="Security_None" linguisticValue="//@valueDomains.0/@subsets.0"/>
    <domain xsi:type="LCPnet:LNodeValue" name="Security_Full" linguisticValue="//@valueDomains.0/@subsets.1"/>
    <domain xsi:type="LCPnet:CNodeValue" name="Security_measured_value" crispValue="0.4"/>
    <linguisticTable name="security_clpt">
      <lines>
        <utility nodeValue="//@nodes.0/@domain.0" utility="//@utilityDomain/@subsets.0"/>
        <utility nodeValue="//@nodes.0/@domain.1" utility="//@utilityDomain/@subsets.4"/>
      </lines>
    </linguisticTable>
  </nodes>
  <nodes xsi:type="LCPnet:LNode" name="bandwidth" outArcs="//@arcs.1 //@arcs.0" valueDomain="//@valueDomains.1">
    <domain xsi:type="LCPnet:LNodeValue" name="Bandwidth_Low" linguisticValue="//@valueDomains.1/@subsets.0"/>
    <domain xsi:type="LCPnet:LNodeValue" name="Bandwidth_Medium" linguisticValue="//@valueDomains.1/@subsets.1"/>
    <domain xsi:type="LCPnet:LNodeValue" name="Bandwidth_High" linguisticValue="//@valueDomains.1/@subsets.2"/>
    <domain xsi:type="LCPnet:CNodeValue" name="Bandwidth_measured_value" crispValue="1.0"/>
    <linguisticTable name="bandwidth_clpt">
      <lines name="bandwidth_clpt_line0">
        <utility name="bandwidth_clpt_line0_lvu0" nodeValue="//@nodes.1/@domain.0" utility="//@utilityDomain/@subsets.0"/>
        <utility name="bandwidth_clpt_line0_lvu1" nodeValue="//@nodes.1/@domain.1" utility="//@utilityDomain/@subsets.2"/>
        <utility name="bandwidth_clpt_line0_lvu2" nodeValue="//@nodes.1/@domain.2" utility="//@utilityDomain/@subsets.4"/>
      </lines>
    </linguisticTable>
  </nodes>
  <nodes xsi:type="LCPnet:LNode" name="resolution" inArcs="//@arcs.1" valueDomain="//@valueDomains.2">
    <domain xsi:type="LCPnet:LNodeValue" name="Resolution_Low" linguisticValue="//@valueDomains.2/@subsets.0"/>
    <domain xsi:type="LCPnet:LNodeValue" name="Resolution_High" linguisticValue="//@valueDomains.2/@subsets.1"/>
    <domain xsi:type="LCPnet:CNodeValue" name="Resolution_measured_value" crispValue="0.51"/>
    <linguisticTable name="resolution_clpt">
      <lines parentValues="//@nodes.1/@domain.0">
        <utility nodeValue="//@nodes.2/@domain.0" utility="//@utilityDomain/@subsets.4"/>
        <utility nodeValue="//@nodes.2/@domain.1" utility="//@utilityDomain/@subsets.0"/>
      </lines>
      <lines parentValues="//@nodes.1/@domain.1">
        <utility nodeValue="//@nodes.2/@domain.0" utility="//@utilityDomain/@subsets.3"/>
        <utility nodeValue="//@nodes.2/@domain.1" utility="//@utilityDomain/@subsets.1"/>
      </lines>
      <lines parentValues="//@nodes.1/@domain.2">
        <utility nodeValue="//@nodes.2/@domain.0" utility="//@utilityDomain/@subsets.0"/>
        <utility nodeValue="//@nodes.2/@domain.1" utility="//@utilityDomain/@subsets.4"/>
      </lines>
    </linguisticTable>
  </nodes>
  <arcs xsi:type="LCPnet:IArc" name="BtoS" startNode="//@nodes.1" endNode="//@nodes.0"/>
  <arcs name="BtoR" startNode="//@nodes.1" endNode="//@nodes.2"/>
  <outcomes xsi:type="LCPnet:COutcome" name="testOutcome" nodeValues="//@nodes.1/@domain.3 //@nodes.2/@domain.2 //@nodes.0/
       @domain.2"/>
  <utilityDomain name="Utility">
    <subsets name="Utility_VL">
      <fuzzySubset y="1.0"/>
      <fuzzySubset x="0.25"/>
    </subsets>
    <subsets name="Utility_L">
      <fuzzySubset/>
      <fuzzySubset x="0.25" y="1.0"/>
      <fuzzySubset x="0.5"/>
    </subsets>
    <subsets name="Utility_M">
      <fuzzySubset x="0.25"/>
      <fuzzySubset x="0.5" y="1.0"/>
      <fuzzySubset x="0.75"/>
    </subsets>
    <subsets name="Utility_H">
      <fuzzySubset x="0.5"/>
      <fuzzySubset x="0.75" y="1.0"/>
      <fuzzySubset x="1.0"/>
    </subsets>
    <subsets name="Utility_VH">
      <fuzzySubset x="0.75"/>
      <fuzzySubset x="1.0" y="1.0"/>
    </subsets>
  </utilityDomain>
  <valueDomains name="Security">
    <subsets name="Security_None">
      <fuzzySubset y="1.0"/>
      <fuzzySubset x="1.0"/>
    </subsets>
    <subsets name="Security_Full">
      <fuzzySubset/>
      <fuzzySubset x="1.0" y="1.0"/>
    </subsets>
  </valueDomains>
  (...)
</LCPnet:LCPnet>
```

Code 6.1 – Sérialisation XML du modèle de préférences.

Seules les nombreuses expressions XPath [Clark et al., 1999] en compliquent légèrement la lecture car elles introduisent des indirections (par exemple *linguisticValue="//@valueDomains.0/@subsets.0"* qui désignent le premier terme linguistique du partitionnement du domaine *Security* : il s'agit de *Security_None*).

6.4 De l'élicitation de LCP-nets à la sélection de services

Les sections suivantes regroupent l'ensemble des étapes effectuées en séquence afin de réaliser la prise de décision de liaison, entre un processus métier et un service, fondée sur les préférences LCP-nets et valeurs courantes de QoS. Cette prise de décision dirigée par les préférences des utilisateurs constitue le cœur de notre approche pour la composition utile de services.

Ainsi, l'un des aspects principaux des LCP-nets réside dans la transposition qui est effectuée, lors de leur compilation, entre un modèle graphique à destination des utilisateurs qui est fondé sur une combinaison des *CP-nets et de l'approche linguistique floue, et une représentation "interne" plus adaptée à la prise de décision sur les préférences. En particulier, les différentes tables d'un LCP-net sont traduites en autant de Systèmes d'Inférence Floue ("Fuzzy Inference System" ou FIS) [Jang, 1993] sur lesquels sera effectué le raisonnement logique. En fait, les tables d'un LCP-net ne sont qu'une des représentations possibles d'un FIS.

L'évaluation d'un LCP-net est donc fondée sur celle des FIS qui lui sont attribués, en utilisant le canevas théorique de la logique floue. La transposition du système initial de représentation permet d'obtenir un niveau de flexibilité, lors de l'évaluation des préférences, comparable à celui disponible lors de leur élicitation. Par exemple, lors de l'évaluation d'un LCP-net, ses valeurs d'entrée peuvent aussi bien être fournies de manière précise ou floue, malgré le strict usage de termes linguistiques (et donc flous) dans ses tables de préférences conditionnelles.

6.4.1 Elicitation des préférences

Tel qu'indiqué précédemment, l'établissement des réseaux de préférences est effectué avant l'exécution du système : un LCP-net ainsi élicité va cependant être en mesure de diriger la sélection des services candidats en fonction de leurs valeurs courantes de QoS à l'exécution. Cette capacité permet ainsi aux utilisateurs d'adapter a priori l'orchestration de services à chaque contexte de déploiement.

Par exemple, afin d'implanter un unique processus de lutte anti-incendie, la sélection dynamique de services peut être ajustée avant exécution en fonction de deux contextes usuels de déploiement auxquels correspondent des LCP-nets distincts : les feux courants d'origine civile, ou la gestion de crises environnementales à plus grande échelle dans laquelle le processus de lutte anti-incendie ne constitue qu'une sous-partie d'un plus grand ensemble de moyens déployés sur le théâtre des opérations. Dans ce dernier cas, des délais d'intervention plus courts pourraient être préférés à un meilleur niveau d'équipement, le théâtre des opérations étant par nature relativement éloigné.

On retrouve cette volonté d'optimisation des délais de décision, et donc d'intervention, dans le modèle de préférences précédemment introduit pour la sélection d'un drone d'imagerie aérienne (cf. figure 6.1). Ce modèle spécifique nous servira alors à illustrer des concepts présentés dans les sections suivantes. Dans ce LCP-net, la sécurité est donnée comme nulle (S_{none}) ou complète (S_{full}), lui est alors attribuée respectivement une valeur d'utilité très faible (*very low*) ou très élevée (*very high*).

La préférence inconditionnelle relative de la bande passante par rapport à la sécurité est matérialisée par un i-arc tendu du nœud B au nœud S. La bande passante est quant à elle partitionnée en trois variables linguistiques B_L (bande passante faible), B_M (bande passante moyenne) et B_H (bande passante élevée) qui couvrent l'ensemble de son domaine de définition. La préférence établie entre ces valeurs est donnée par la CPT attachée au nœud B : elle exprime ainsi une préférence très faible (*very low*) pour une bande passante faible, moyenne (*medium*) pour une bande passante évaluée elle aussi comme moyenne, et finalement très élevée (*very high*) pour une bande passante élevée.

La résolution est elle aussi partitionnée par l'utilisation de deux variables linguistiques : R_L (résolution faible) et R_H (résolution haute). La préférence entre ces valeurs est par contre conditionnée par celle prise par la bande passante : cette préférence conditionnelle est figurée par un cp-arc établi entre les nœuds B et R. Ainsi, si la bande passante est faible, une résolution faible est fortement préférée ; si la bande passante est moyenne, la résolution moyenne dispose d'une plus grande utilité ; et si la bande passante est élevée, une résolution élevée est fortement préférée.

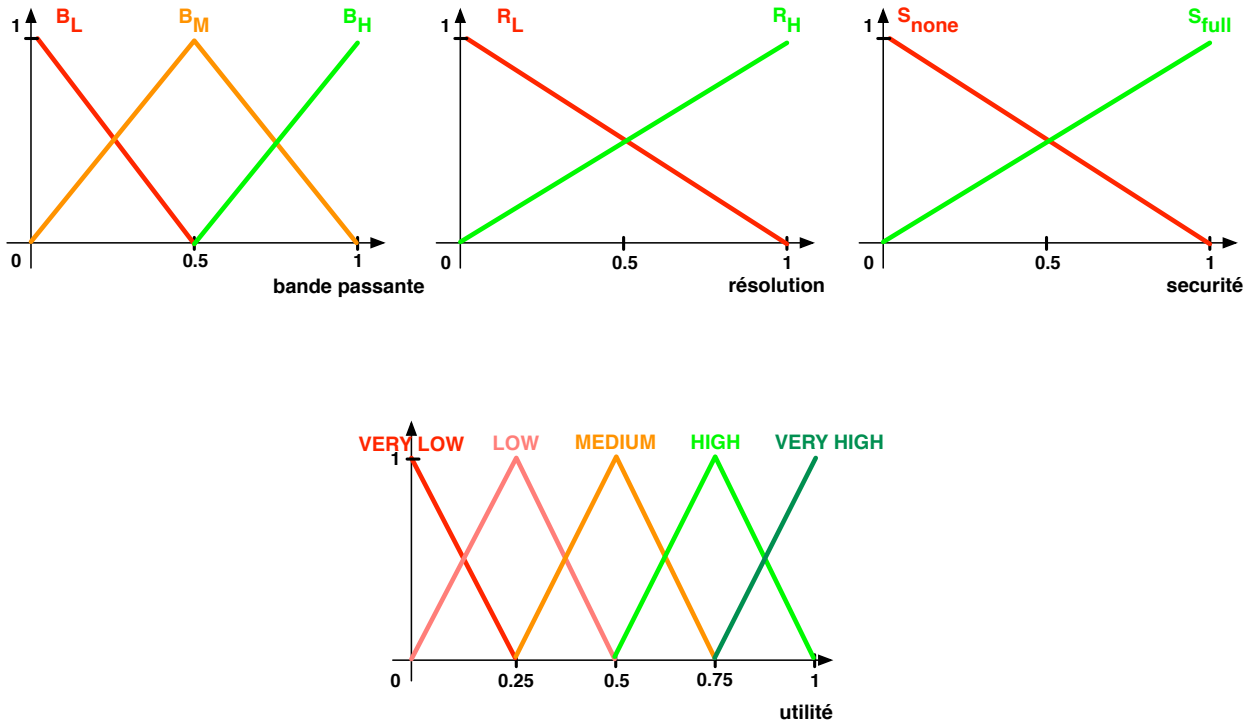

FIGURE 6.4 – Partitionnements flous de la *bande passante*, *résolution*, *sécurité* et *utilité*.

La sémantique des termes linguistiques utilisés dans les tables de préférences, que ce soit sur la sécurité, la bande passante ou la résolution d'image, est donnée au préalable par le partitionnement flou indiqué dans la figure 6.4. A l'issue de ce partitionnement flou, de multiples sous-ensembles flous sont obtenues (B_L, R_H, *low*, *very high*, etc.).

Après élicitation, un modèle de préférences est donc obtenu dans son intégralité. Dans le modèle spécifique que nous avons précédemment figuré, et qui servira de base pour l'illustration des prochaines étapes menant à une prise de décision de liaison entre processus et service, nous avons pu voir que la bande passante est toujours préférée à la sécurité par l'utilisateur, et que si la bande

```
FIS traductionNoeudVersFIS(noeud origine) {
        // création d'un nouveau FIS
        FIS fis = new FIS();

        // I - création des variables d'entrée du FIS

        pour chaque(noeud d'entrée de la CPT, y compris le noeud d'origine) {
                1 - création d'une variable réelle correspondante;
                2 - création du domaine de définition de la variable à partir des
                    partitionnements flous;
                        2.1 - conversion de chaque SEF en fonction d'appartenance;
                3 - ajout de la variable d'entrée au fis
        }

        // II - création de la variable de sortie du FIS à partir de la définition de l'
            utilité

        1 - création d'une variable réelle correspondante;
        2 - création du domaine de définition de la variable à partir des partitionnements
            flous de l'utilité;
                2.1 - conversion de chaque SEF en fonction d'appartenance;
        3 - ajout de la variable de sortie au fis

        /// III - création d'une règle floue par cellule dans la CPT

        // création d'un ensemble vide de règles floues (un par noeud)
        FuzzyRuleSet ruleSet = new FuzzyRuleSet();

        pour chaque(ligne de la CPT du noeud) {
                pour chaque(cellule de la ligne) {
                        1 - création d'une règle floue
                        2 - déduction des 'antécédents' de la règle à partir des valeurs des
                             noeuds d'entrée
                        3 - déduction du 'conséquent' de la règle à partir de la valeur de
                            la cellule
                        4 - ajout de la règle à l'ensemble ruleSet
                }
        }

        return fis.addFuzzyRuleSet(ruleSet);
}
```

Code 6.2 – Pseudo-code de l'algorithme de traduction de CPT.

passante est faible, des images de basse résolution sont préférées car il souhaite que les images soient obtenues le plus rapidement possible.

6.4.2 Traduction des préférences

Par une utilisation habile des outils mis à notre disposition dans le domaine de la logique floue, il est possible de traduire automatiquement les modèles graphiques de préférence LCP-net précédemment élicités vers une représentation bien plus adaptée pour mener une prise de décision efficace lors de l'exécution du système.

Cette utilisation se traduit par la mise au point d'un algorithme permettant d'obtenir de manière automatique les ensemble de règles floues des FIS qui correspondent à chaque table d'utilité d'un modèle : les FIS ainsi obtenus sont "locaux" aux nœuds du réseau auxquels ils sont rattachés, et

sont donc intimement liés chacune de leurs dimensions de QoS. Il s'agit d'un procédé semblable à celui utilisé que dans le domaine du contrôle flou [Driankov et al., 1993, Castro, 1995]. A l'exécution, les entrées de ces FIS fraîchement obtenus pourront être des valeurs précises (et donc numériques) ou floues (par exemple sous forme de sous-ensembles flous), obtenues des services supervisés. Le pseudo-code de cet algorithme de traduction est donné par l'extrait de code 6.2.

Le même processus de traduction peut être appliqué à tous les nœuds du modèle, qu'ils soient conditionnels ou non. Ainsi, à partir des CPT rattachés aux nœuds R, B, S et des partitionnements flous des domaines de *résolution*, *bande passante*, *sécurité* et *utilité*, nous obtenons un FIS spécifique pour chaque nœud après application de notre algorithme (cf. codes 6.3, 6.4 et 6.5) : ces FIS sont ici représentés dans le langage FCL [IEC, 2001] utilisé en interne par notre canevas de décision sur LCP-nets dont l'implantation sera présenté au chapitre 9.

6.4.3 Evaluation des préférences

Suite à la traduction préalable d'un modèle de préférences vers un mode de représentation plus adapté pour mener à bien la prise de décision, son évaluation peut être effectuée lorsque les données d'entrée nécessaires sont disponibles. Cette évaluation est scindée, ci-dessous, en quatre étapes clés permettant d'obtenir l'utilité de chaque service supervisé à partir de ses valeurs courantes de QoS : l'*injection des valeurs de QoS des services*, l'*inférence des utilités locales pour chaque service*, et finalement le *calcul de l'utilité globale de chaque service*, assisté par le *calcul du poids des nœuds* des préférences.

Par ailleurs, si le concept de liaison tardive précédemment présenté dans le chapitre 5 suppose que l'injection des valeurs est effectuée dynamiquement au cours de l'orchestration de services sur la base de la QoS courante des services, le principe d'évaluation des préférences ici-présenté est indépendant de cette mise en œuvre particulière. En tant que telle, l'injection des valeurs peut avoir lieu à tout moment après la traduction des préférences.

Injection des valeurs de QoS des services

La première étape de l'évaluation des préférences se concrétise lorsque de multiples valeurs de Qualité de Service obtenues à partir d'un canevas externe de supervision, tel qu'abordé dans le chapitre 5, sont injectées dans l'ensemble de FIS obtenu par traduction d'un modèle de préférences, dans le but ultime d'obtenir une quantification de l'utilité globale de chaque service candidat en sortie.

Ces valeurs peuvent être de deux types : valeurs précises de QoS *que l'on normalise sur l'échelle [0,1]*, ou valeurs floues qui peuvent nécessiter des "ajustements" *via* une normalisation de domaine.

Pour l'exemple qui nous sert de fil conducteur, nous nous focalisons sur l'offre non-fonctionnelle courante d'un service S_1 dans le but d'obtenir son utilité globale, un processus similaire pouvant être appliqué à tout autre service offrant les mêmes dimensions de QoS . A un instant t les valeurs (B', S_{full}, R') ont été mesurées pour ce service, ce triplet constitue l'offre non-fonctionnelle *courante* du service.

B' correspond à une valeur précise de bande passante évaluée à 30 Ko/s, cette valeur est ensuite normalisée sur [0,1] en 0.30. Si on se rapporte au partitionnement flou du domaine de la bande

```
FUNCTION_BLOCK fbName

VAR_INPUT
        B : REAL;
        R : REAL;
END_VAR

VAR_OUTPUT
        Utility : REAL;
END_VAR

FUZZIFY B
        TERM Bandwidth_High :=  (0.5, 0.0) (1.0, 1.0) ;
        TERM Bandwidth_Low :=  (0.0, 1.0) (0.5, 0.0) ;
        TERM Bandwidth_Medium :=  (0.0, 0.0) (0.5, 1.0) (1.0, 0.0) ;
END_FUZZIFY

FUZZIFY R
        TERM Resolution_High :=  (0.0, 0.0) (1.0, 1.0) ;
        TERM Resolution_Low :=  (0.0, 1.0) (1.0, 0.0) ;
END_FUZZIFY

DEFUZZIFY Utility
        TERM Utility_H :=  (0.5, 0.0) (0.75, 1.0) (1.0, 0.0) ;
        TERM Utility_L :=  (0.0, 0.0) (0.25, 1.0) (0.5, 0.0) ;
        TERM Utility_M :=  (0.25, 0.0) (0.5, 1.0) (0.75, 0.0) ;
        TERM Utility_VH :=  (0.75, 0.0) (1.0, 1.0) ;
        TERM Utility_VL :=  (0.0, 1.0) (0.25, 0.0) ;
        ACCU : MAX;
        METHOD : COG;
        DEFAULT := 0.0;
        RANGE := (0.0 .. 1.0);
END_DEFUZZIFY

RULEBLOCK Rules
        ACT : MIN;
        AND : MIN;
        RULE 1 : IF (B is Bandwidth_Low) and (R is Resolution_Low) THEN Utility is
            Utility_VH;
        RULE 2 : IF (B is Bandwidth_Low) and (R is Resolution_High) THEN Utility is
            Utility_VL;
        RULE 3 : IF (B is Bandwidth_Medium) and (R is Resolution_Low) THEN Utility is
            Utility_H;
        RULE 4 : IF (B is Bandwidth_Medium) and (R is Resolution_High) THEN Utility is
            Utility_L;
        RULE 5 : IF (B is Bandwidth_High) and (R is Resolution_Low) THEN Utility is
            Utility_VL;
        RULE 6 : IF (B is Bandwidth_High) and (R is Resolution_High) THEN Utility is
            Utility_VH;
END_RULEBLOCK

END_FUNCTION_BLOCK
```

Code 6.3 – FIS obtenu par traduction du nœud R.

```
FUNCTION_BLOCK fbName

VAR_INPUT
        B : REAL;
END_VAR

VAR_OUTPUT
        Utility : REAL;
END_VAR

FUZZIFY B
        TERM Bandwidth_High :=  (0.5, 0.0) (1.0, 1.0) ;
        TERM Bandwidth_Low :=  (0.0, 1.0) (0.5, 0.0) ;
        TERM Bandwidth_Medium :=  (0.0, 0.0) (0.5, 1.0) (1.0, 0.0) ;
END_FUZZIFY

DEFUZZIFY Utility
        TERM Utility_H :=  (0.5, 0.0) (0.75, 1.0) (1.0, 0.0) ;
        TERM Utility_L :=  (0.0, 0.0) (0.25, 1.0) (0.5, 0.0) ;
        TERM Utility_M :=  (0.25, 0.0) (0.5, 1.0) (0.75, 0.0) ;
        TERM Utility_VH :=  (0.75, 0.0) (1.0, 1.0) ;
        TERM Utility_VL :=  (0.0, 1.0) (0.25, 0.0) ;
        ACCU : MAX;
        METHOD : COG;
        DEFAULT := 0.0;
        RANGE := (0.0 .. 1.0);
END_DEFUZZIFY

RULEBLOCK Rules
        ACT : MIN;
        AND : MIN;
        RULE 1 : IF B is Bandwidth_Low THEN Utility is Utility_VL;
        RULE 2 : IF B is Bandwidth_Medium THEN Utility is Utility_M;
        RULE 3 : IF B is Bandwidth_High THEN Utility is Utility_VH;
END_RULEBLOCK

END_FUNCTION_BLOCK
```

Code 6.4 – FIS obtenu par traduction du nœud B.

```
FUNCTION_BLOCK fbName

VAR_INPUT
        S : REAL;
END_VAR

VAR_OUTPUT
        Utility : REAL;
END_VAR

FUZZIFY S
        TERM Security_Full :=  (0.0, 0.0) (1.0, 1.0) ;
        TERM Security_None :=  (0.0, 1.0) (1.0, 0.0) ;
END_FUZZIFY

DEFUZZIFY Utility
        TERM Utility_H :=  (0.5, 0.0) (0.75, 1.0) (1.0, 0.0) ;
        TERM Utility_L :=  (0.0, 0.0) (0.25, 1.0) (0.5, 0.0) ;
        TERM Utility_M :=  (0.25, 0.0) (0.5, 1.0) (0.75, 0.0) ;
        TERM Utility_VH :=  (0.75, 0.0) (1.0, 1.0) ;
        TERM Utility_VL :=  (0.0, 1.0) (0.25, 0.0) ;
        ACCU : MAX;
        METHOD : COG;
        DEFAULT := 0.0;
        RANGE := (0.0 .. 1.0);
END_DEFUZZIFY

RULEBLOCK Rules
        ACT : MIN;
        AND : MIN;
        RULE 1 : IF S is Security_None THEN Utility is Utility_VL;
        RULE 2 : IF S is Security_Full THEN Utility is Utility_VH;
END_RULEBLOCK

END_FUNCTION_BLOCK
```

Code 6.5 – FIS obtenu par traduction du nœud S.

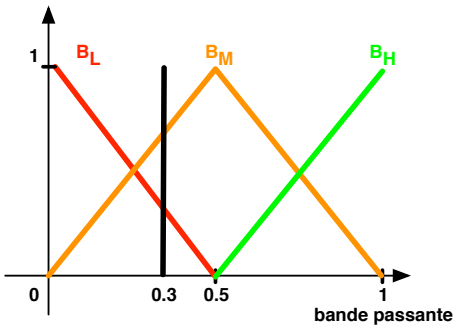

FIGURE 6.5 – Valeur précise de bande passante, sous forme de singleton.

passante, on voit que B' peut alors être représentée comme un singleton qui intersecte les sous-ensembles flous B_L et B_M, tel que dépeint sur la figure 6.5.

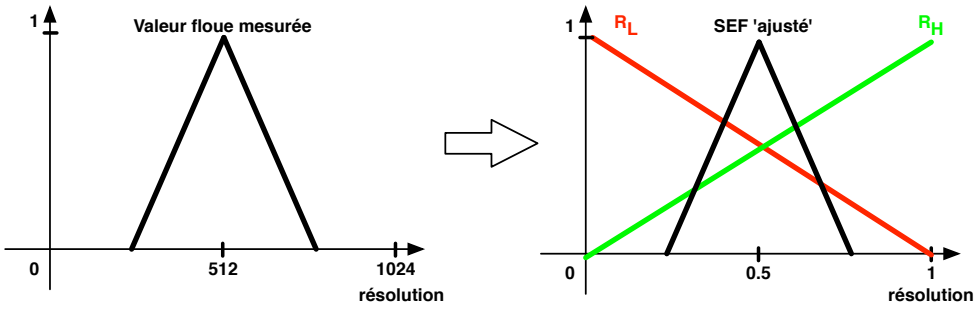

FIGURE 6.6 – Valeur floue de résolution, sous forme de SEF ajusté au domaine.

R' correspond à une valeur floue de résolution qui doit d'abord être "ajustée" par une normalisation de domaine sur l'échelle [0,1], tel que représenté sur la figure 6.6, avant son injection dans le système.

Inférence des utilités locales pour chaque service

Une fois les valeurs de QoS injectées, l'inférence des utilités locales à chaque nœud du modèle peut être effectuée. Cette inférence est ici réalisée par le biais du Modus Ponens Généralisé (cf. section 3.1.2).

Puisque nous disposons d'une valeur B' de bande passante, il est possible de continuer à dérouler notre exemple en vue d'obtenir l'utilité locale du nœud B. En effet, comme son FIS nous l'indique, seul une entrée pour la variable B est requise, la préférence sur la bande passante n'étant définie conditionnellement à aucune autre propriété de Qualité de Service.

Une trace visuelle du processus d'inférence floue réalisée par le biais du MPG, sur la base des informations et paramètres fournis par le FIS et la valeur B', est donnée sur la figure 6.7. Cette inférence utilise l'implication de Mamdani [Mamdani et Assilian, 1975] et la T-Norme de Zadeh comme opérateurs flous. La figure montre que les deux points d'intersection du singleton B' avec les sous-ensembles flous B_L et B_M du domaine d'entrée sont projetés sur les SEF correspondants du domaine de sortie, tel qu'indiqué dans les règles obtenues par traduction de la table du nœud

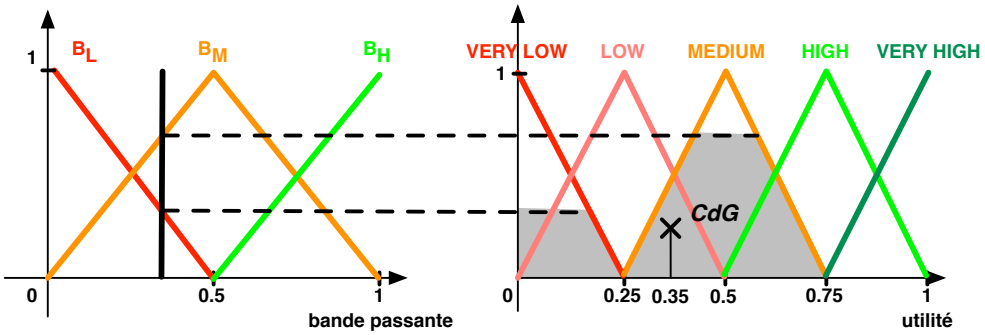

FIGURE 6.7 – Inférence floue, de la bande passante mesurée à l'utilité locale de B.

B, et ce dans le but d'obtenir un nouveau sous-ensemble flou (représenté par la forme géométrique grise) qui sera defuzzifié en calculant son centre de gravité (CdG). La valeur d'utilité locale 0.35 sur l'échelle [0,1] est alors obtenue en sortie du processus d'inférence. Si on souhaite obtenir un mot en sortie au lieu d'une valeur numérique, et cela sans perte de précision, il est possible d'utiliser un formalisme à base de 2-tuple (cf. section 3.1.3). Le même processus sera appliqué pour R et S à partir de l'offre non fonctionnelle de S_1 dans le but d'obtenir leurs utilités locales respectives.

Tels que nous les voyons ici, les opérateurs d'inférence floue que nous avons mis en œuvre pour procéder à l'obtention des utilités locales sont prédéterminés. Cependant, un certain niveau de contrôle sur le choix de ces opérateurs pourrait être donné aux utilisateurs, en fonction de leur rôle dans le système, dans l'optique de permettre l'adaptation du processus d'inférence à chaque contexte spécifique de déploiement des préférences (par exemple au niveau de sa performance ou de sa précision). Il faut cependant disposer d'un solide niveau de connaissance sur les différents opérateurs existants pour réaliser un choix éclairé.

Par ailleurs, si le mécanisme d'inférence ici-présenté consomme des valeurs numériques précises et produit des utilités locales sous forme réelle, une alternative serait d'effectuer une inférence linguistique "de bout en bout" : un processus entièrement linguistique pourrait en effet être en adéquation plus fine avec les intentions initiales de l'utilisateur qui ont été, rappelons le, retranscrites de manière linguistique lors de l'expression des LCP-nets. Les 2-tuples précédemment introduits dans la section 3.1 permettent justement un traitement automatique des déclarations linguistiques théoriquement sans perte d'information. Si l'on considère alors les valeurs des propriétés de QoS des services comme fournies sous la forme de 2-tuples, un mécanisme *ad hoc* d'inférence [Alcalá et al., 2007] pourrait être étudié et implanté. Il nécessiterait notamment l'utilisation de plusieurs opérateurs spécifiques d'agrégation linguistique [Xu, 2008], fondés sur les opérateurs OWA de Yager [Yager, 1988]. Cette alternative ne sera pas présentée dans le cadre de ce manuscrit de thèse.

Calcul de l'utilité globale de chaque service

Pour finir, au cours du calcul de l'utilité globale d'un service, l'agrégation des multiples utilités locales précédemment inférées est effectuée. Cependant, il est nécessaire de prendre en compte le fait que les arcs dans les modèles de préférences originels indiquent l'importance relative des nœuds (et donc des propriétés de QoS) qu'ils interconnectent. Cette importance relative implicite doit ainsi être répercutée sur chaque utilité locale pour l'obtention de l'utilité globale d'un service. Par exemple,

dans notre précédent modèle, la bande passante est plus importante que la sécurité et la résolution, car elle se situe au "sommet" du graphe de préférence. S et R partagent quant à eux le même degré d'importance car ils sont situés à la même profondeur dans le graphe.

Par ailleurs, si la nature numérique des utilités spécifiés dans les tables des UCP-Nets incite les utilisateurs à exprimer directement l'importance relative des propriétés de QoS en ajustant directement l'ordre de grandeur des valeurs d'utilités dans les différentes tables, on ne dispose plus de cette possibilité avec les termes linguistiques utilisés lors de modélisation des LCP-nets. Ainsi, afin de préserver l'importance relative des nœuds des LCP-nets lors du calcul de l'utilité globale, des poids sont calculés et associés à chaque nœud du graphe de préférence.

Cette quatrième étape clé de calcul du poids des nœuds est donc intégrée au sein même du calcul de l'utilité globale. Leur obtention est en fait directement corrélée à la profondeur des nœuds dans le graphe. Ces poids sont valués sur [0,1] et leur distribution obtenue *via* une fonction spécialement définie *a minima* sur [0, *profondeur max du graphe*] : si on se réfère à la sémantique du formalisme LCP-net, plus la profondeur est faible, plus le poids obtenu par la fonction doit être élevé. L'ensemble des poids obtenus est ensuite normalisé afin de totaliser 1, dans le but d'effectuer une moyenne pondérée. Dans ce cas d'utilisation, nous fixons à l'avance une fonction de distribution des poids définie sur [0,100] :

$$g(x) = 1/x^2 + 0.8$$

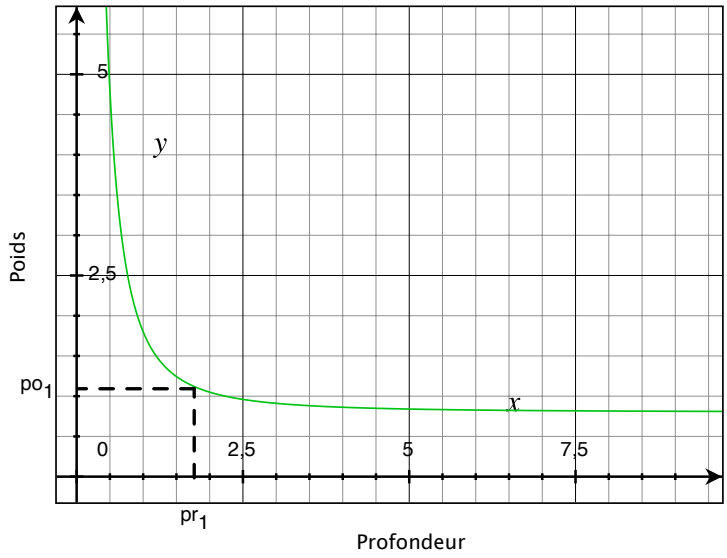

FIGURE 6.8 – La fonction g de distribution des poids choisie pour cet exemple.

A partir de cette fonction (cf. son profil donné par la figure 6.8), et de notre LCP-net d'exemple, nous obtenons les paires (*profondeur*, *poids*) suivantes après normalisation : (1, 0.529) et (2, 0.235). Ces paires sont ensuite associées à chaque nœud du modèle de préférences (cf. figure 6.9).

L'utilité globale d'un service est ensuite calculée en agrégeant l'ensemble des utilités locales *via* un opérateur de référence Δ intégrant les poids préalablement obtenus au sein d'une simple moyenne

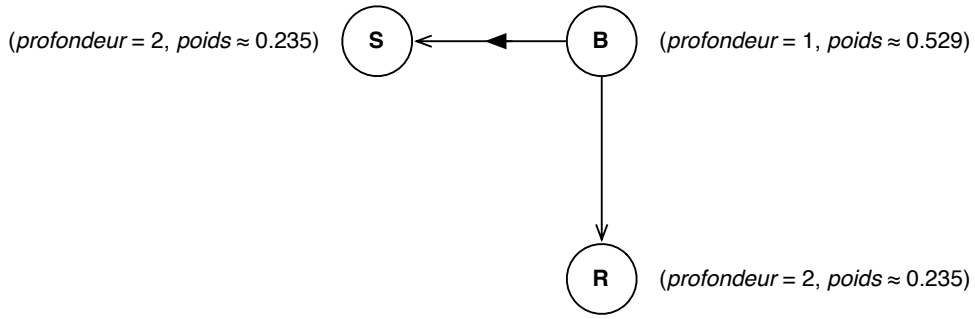

FIGURE 6.9 – Poids des nœuds obtenus à partir de leur profondeur dans le graphe.

pondérée. Dans notre exemple, les valeurs (B', S_{full}, R') ont été mesurées pour un service S_1 donné à un instant t, les utilités locales obtenues pour chacune de ces dimensions de QoS sont les suivantes : utilitéLocale$(B') = 0.35$ (tel que démontré ci-dessus), utilitéLocale$(S_{full}) = 1$, utilitéLocale$(B', R') = 0.20$. L'opérateur Δ est alors appliqué comme suit à partir des poids précédemment calculés :

$$\Delta(B', S_{full}, R') = 0.529 \times 0.35 + 0.235 \times 1 + 0.235 \times 0.20 \approx 0.47$$

L'utilité globale ainsi obtenue correspond à l'offre non-fonctionnelle courante de S_1, elle est évaluée à 0.47 sur l'échelle [0,1].

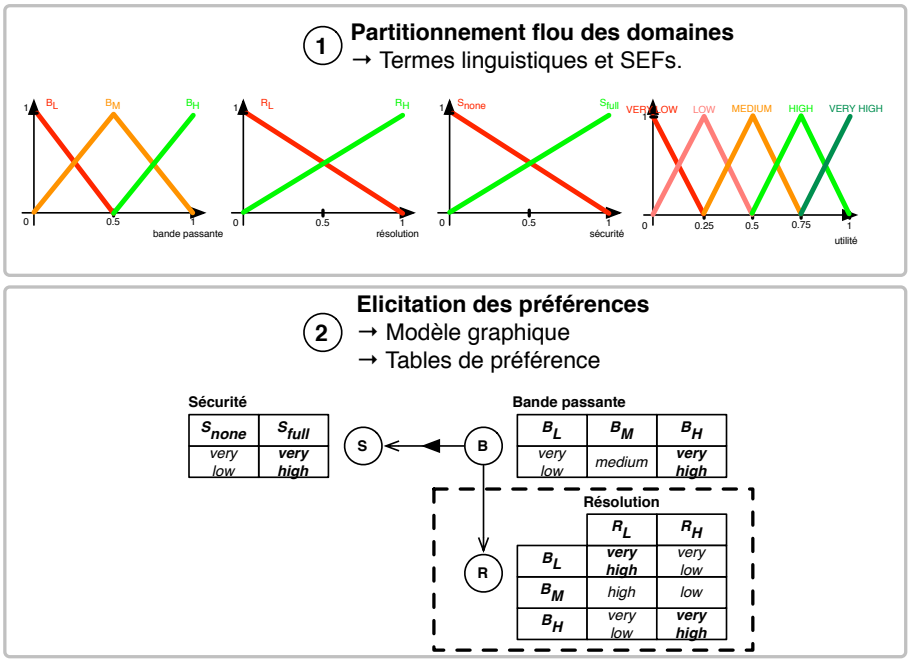

FIGURE 6.10 – Résumé des étapes engagées pour l'élicitation d'un LCP-net globale d'un service.

La figure 6.10 résume l'ensemble des étapes qui ont été engagées pour l'élicitation du partitionnement des domaines et celle des préférences, tandis que la figure 6.11 illustre le processus d'obtention de l'utilité globale du service S_1 à partir des préférences élicitées, en se focalisant sur le nœud R.

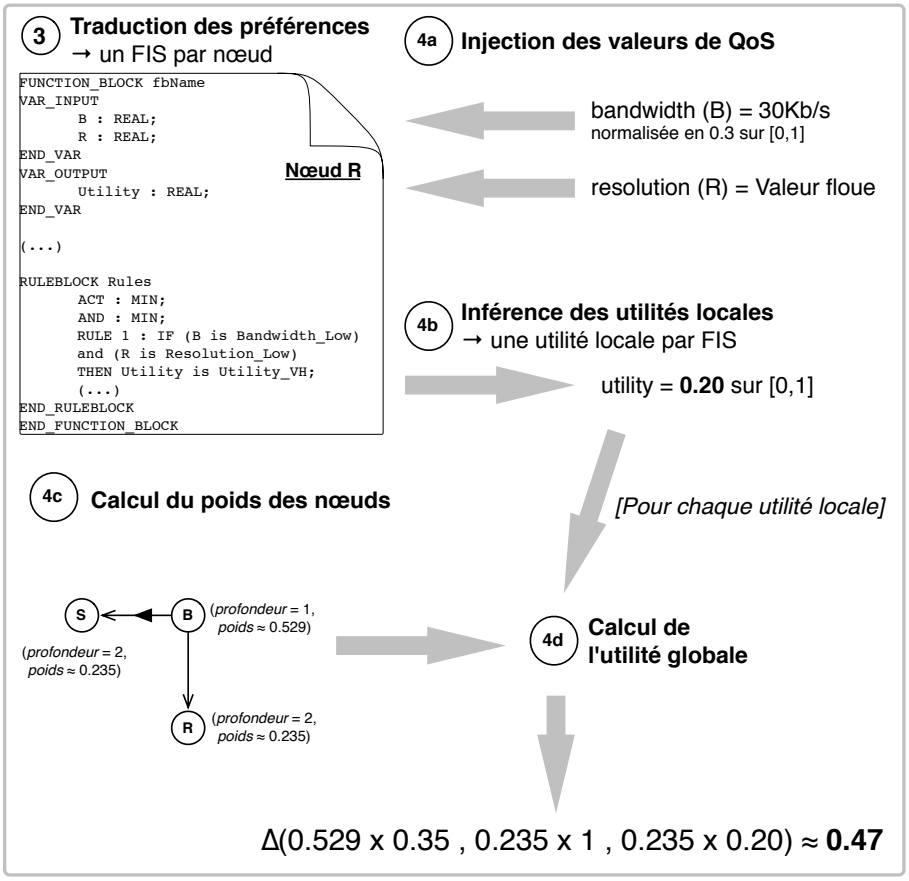

FIGURE 6.11 – Résumé des étapes engagées pour l'obtention de l'utilité globale d'un service.

6.4.4 Sélection d'un service : indécision levée par l'usage d'un LCP-net

L'ultime palier, après avoir élicité, traduit et évalué les préférences au regard des différentes offres non-fonctionnelles courantes des services candidats, consiste à comparer leurs différentes utilités globales de manière à obtenir un ordre total sur les services, et d'en sélectionner le premier (donc celui à l'utilité maximale) comme résultat du processus global de décision.

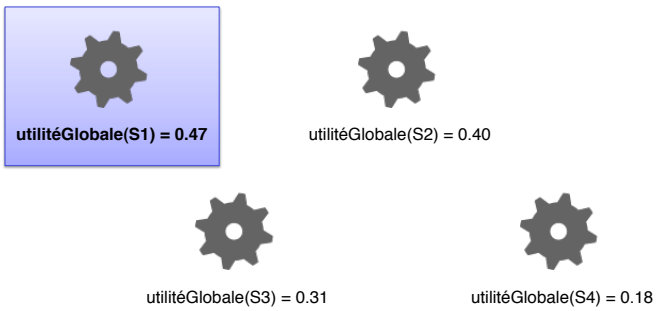

FIGURE 6.12 – Comparaison des services candidats sur la base de leur utilité globale.

Dans les étapes précédentes, nous avons détaillé le calcul de l'utilité globale d'un service S_1. Le même procédé a été appliqué à trois autres services S_2, S_3 et S_4 fonctionnellement et non-fonctionnellement équivalents à S_1. A partir de l'utilité globale de chacun de ces services calculée sur la base de leurs offres respectives courantes sur B, S et R, il est possible de sélectionner S_1 comme service offrant l'utilité maximale (cf. figure 6.12).

Intérêt patent de l'approche linguistique floue lors de la sélection

L'approche linguistique floue mise en œuvre par les LCP-nets va alors permettre, dans certains cas de figure, d'effectuer une décision de sélection entre deux services là où, paradoxalement, l'approche "crisp" (précise, par opposition à "floue") utilisée par les autres membres de la famille *CP-nets resterait muette.

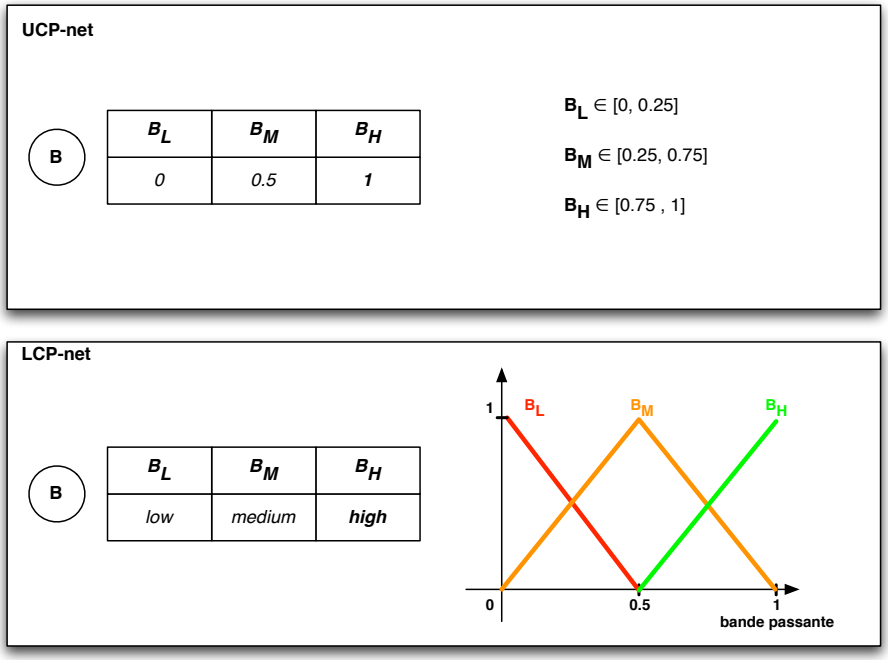

FIGURE 6.13 – Deux modélisations pour une même préférence.

Pour illustrer cette affirmation, on compare l'évolution de l'utilité obtenue à l'issue du processus d'évaluation d'un LCP-net à une dimension (pour plus de simplicité, car *utilité locale* et *globale* sont alors confondues) à celle obtenue en sortie pour un UCP-net équivalent, en fonction de leur unique valeur d'entrée. Les deux réseaux de préférences modélisent l'affirmation *"l'utilité d'un service est d'autant plus élevée que sa bande passante est élevée"*, cependant, le LCP-net est accompagné d'un partitionnement flou par trois SEF du domaine de la bande passante, là où l'UCP-net considère un découpage strict par trois intervalles distincts (cf. figure 6.13).

L'utilité d'un service en fonction de sa bande passante, calculée à partir de l'UCP-net est donnée par la figure 6.14. On remarque la forme "en escalier" de la courbe d'utilité, elle correspond à la traduction directe sur le graphe de la table d'utilité "crisp" de l'UCP-net ainsi que du partitionnement de la bande passante qui l'accompagne. Par conséquent, on distingue trois zones pour lesquelles des

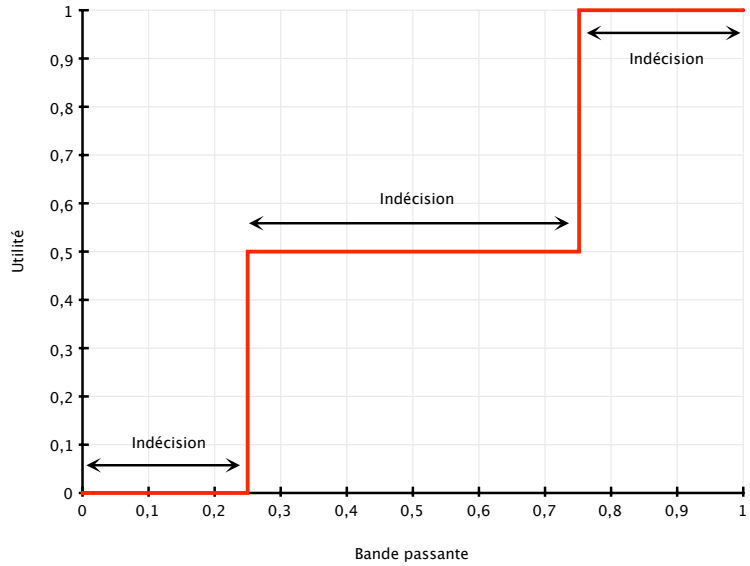

FIGURE 6.14 – UCP-net : Utilité d'un service en fonction de sa bande passante.

services offrants des bandes passantes proches mais différentes seront considérés comme équivalents (même utilité) et où la décision n'est donc plus possible.

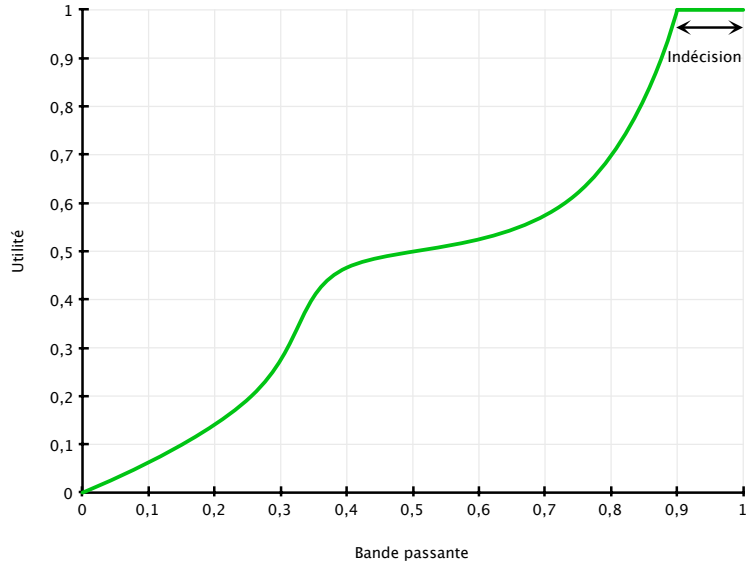

FIGURE 6.15 – LCP-net : Utilité d'un service en fonction de sa bande passante.

A contrario, on constate que les zones d'indécision sont particulièrement réduites (il n'en reste plus qu'une, de taille inférieure) en sortie de l'évaluation du LCP-net 6.15. La courbe d'utilité a été lissée par le partitionnement flou du domaine d'entrée ainsi que le processus d'évaluation à base de règles floues issue de la traduction de l'unique CPT du réseau de préférences (cf. section 6.4.2).

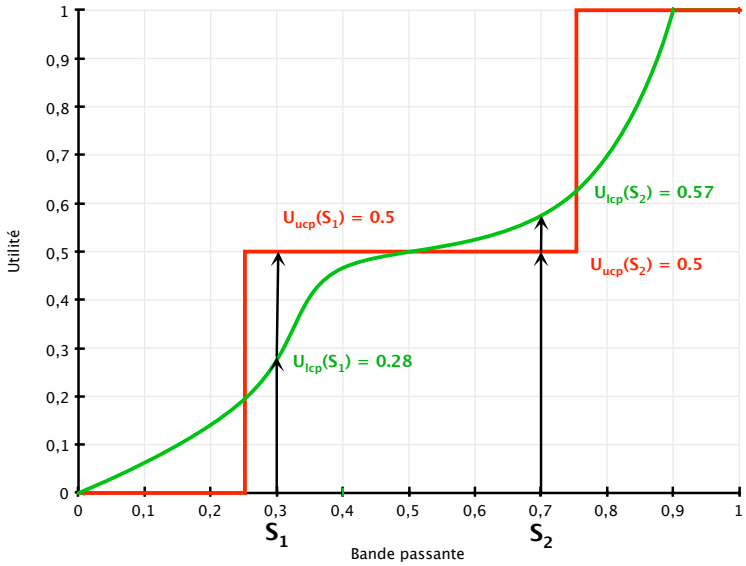

FIGURE 6.16 – UCP-net *vs.* LCP-net : différences de précision.

Pour deux services S_1 et S_2 offrant respectivement une bande passante courante de 0.3 et 0.7 sur l'ensemble $[0, 1]$, on obtient deux valeurs d'utilité distinctes (0.28 et 0.57) à l'issue du processus d'inférence floue sur le LCP-net, il est donc possible de choisir le service S_2 car son utilité est la plus élevée. *A contrario*, deux utilités identiques évaluées à 0.5 sont obtenues par l'UCP-net, il n'est alors pas possible de déterminer le service le plus utile. Cette comparaison est représentée sur la figure 6.16.

6.5 Vers un traitement entièrement linguistique des préférences

Dans le cas d'une mise en œuvre entièrement automatique du processus de décision telle que présentée dans ce chapitre, l'obtention de valeurs d'utilités locales et globales numériques est pertinente car les outils d'agrégation et de comparaison habituels s'avèrent alors adaptés et efficaces. Cependant, dans d'autres contextes d'application la prise de décision finale pourrait reposer sur l'intervention humaine. Dans ce second cas, une méthode d'inférence et d'agrégation entièrement linguistique, par exemple à base de 2-tuples flous, permettrait de faire "remonter" au preneur de décision des informations sous une forme qualitative plus intelligible qu'un ensemble de valeurs numériques réelles, et cela sans perte de précision.

En se fondant sur le procédé représenté dans sa globalité par les figures 6.10 et 6.11, une partie des perfectionnements liés à un traitement linguistique "de bout en bout" des préférences qui ont été évoqués précédemment, pourrait être implantée comme suit :

– les partitionnements flous des domaines figurés à l'étape 1, y compris celui de l'utilité, entraînerait la production de termes linguistiques sous la forme de 2-tuples, *i.e.* des paires (s_i, α), avec $\alpha = 0$, c'est-à-dire dire sans déplacement latéral. Par exemple, la bande passante serait représentée, en gardant le partitionnement flou présenté sur la figure 6.4, par $\{(low,0)\,;\ (medium,0)\,;\ (high,0)\}$;

- à l'étape 2, les tables de préférence contiendraient elles-aussi des 2-tuples (par exemple (*very low*,0) ou (*very low*,−0.2)), au lieu de simples termes linguistiques (par exemple *very low*). En effet, l'utilisateur exprime ses préférences de manière graphique, c'est pourquoi un déplacement latéral des termes linguistiques originaux pourrait apparaître. Par la suite, les règles obtenues par traduction des préférences en FIS pourraient posséder des 2-tuples : par exemple "If the resolution is (*low*,0) and the bandwidth is (*high*,0) then the utility is (*very low*,−0.2)" ;
- à l'étape 3, on procéderait à la traduction des préférences vers des systèmes d'inférence floue (FIS) pour 2-tuples, tel que proposé par Alcalá *et al.* [Alcalá et al., 2007] ;
- lors de l'étape 4a, les valeurs injectées le seraient sous forme de 2-tuples ;
- l'étape 4b aurait pour résultat l'inférence d'utilités locales sous forme de 2-tuples ;
- l'étape 4c de calcul des poids serait gardée en l'état ;
- pour finir, l'étape 4d de calcul de l'utilité globale procéderait à l'agrégation de 2-tuples pondérés [Herrera et Martínez, 2000] de manière à obtenir une utilité globale elle aussi sous forme de 2-tuple.

Il est aussi intéressant de noter que les 2-tuples flous pourraient s'avérer particulièrement utiles dans le cas où l'on souhaiterait offrir à l'utilisateur la possibilité de modifier les CPT des préférences *a posteriori*, par l'ajout ou la suppression d'une colonne ou d'une ligne : le système procéderait alors à la mise à jour et calcul de nouvelles utilités dans les tables, ces utilités pourraient alors être exprimées avec un déplacement latéral (par exemple sous la forme (*very high*,−0.1)) quand bien même les utilités initiales des tables seraient sous forme $(s_i, 0)$. En fait, dès que l'on souhaite procéder à une reconfiguration automatique d'un LCP-net, on doit changer la granularité des domaines considérés et les 2-tuples apparaissent comme un outil de choix pour y procéder, en permettant au système de conserver une relation avec l'ensemble de termes originaux (en l'occurrence grâce aux s_i des 2-tuples).

6.6 Conclusion

Afin de permettre la sélection de services dans le contexte multi-critères dans lequel nous nous trouvons, des préférences établies entre les propriétés non-fonctionnelles des services doivent être élicitées afin d'obtenir un ordre total sur les candidats. A cette fin, nous avons mis en avant dans ce chapitre la contribution scientifique de cette thèse portant sur l'établissement d'une nouvelle variante des CP-nets, appelée LCP-nets, qui combine certaines propriétés des UCP-nets et TCP-nets avec les avantages apportés par une approche linguistique floue pour discrétiser les domaines continus des variables. Un LCP-net permet par ailleurs l'expression de l'*utilité qualitative* des affectations de ses variables dans des tables de préférences conditionnelles dédiées.

Les LCP-nets s'avèrent ainsi particulièrement adaptés au contexte des SOA Sémantiques puisqu'ils permettent aux utilisateurs de partitionner efficacement les domaines continus des variables de Qualité de Service avec des termes linguistiques appropriés, ainsi que d'exprimer l'utilité des affectations d'une manière qualitative plutôt que numérique, cette dernière s'avérant souvent artificielle. L'exécution du canevas de support des LCP-nets dans ce contexte permettra alors de comparer les services candidats entre eux, sur la base des utilités globales obtenues. C'est donc en ce sens que

l'on considère les LCP-nets comme la pierre angulaire de notre approche pour la *composition utile de services*.

Nous avons supposé pour le moment que les utilités des différents LCP-nets sont toujours exprimées en utilisant le même ensemble de termes linguistiques (cf. le partitionnement de l'utilité ("utility") sur la figure 6.4), mais cette restriction pourrait être levée par la mise en œuvre d'un procédé entièrement linguistique fondée sur les 2-tuples par l'utilisation d'une approche multi-granulaire [Herrera et al., 2002] lors du calcul de la fonction d'utilité globale. De même, l'hypothèse sur le positionnement des termes linguistiques (centrés sur 0.5, les termes étant alors distribués de manière équidistante) pourrait elle aussi être supprimée en implantant des approches permettant de faire face à des ensembles de termes déséquilibrés ("unbalanced term sets") [Herrera et al., 2008].

Il est aussi important de souligner que bien qu'établis originellement pour le contexte particulier des SSOA, et présentés comme tel dans ce chapitre, le formalisme LCP-net et son canevas d'inférence *ne sont pas liés à un domaine applicatif spécifique* et pourraient tout à fait être déployés et appliqués à d'autres contextes de décision multi-critères exhibant des contraintes similaires. Citons à titre d'exemple les cas d'utilisation proposés dans la littérature pour la présentation d'imagerie médicale [Boutilier et al., 2004] et la planification d'un voyage en avion [Brafman et Domshlak, 2002]. Pour le second, les préférences qui étaient initialement portées dans les tables de manière très stricte, car fondées sur la relation d'ordre binaire '>' des CP-nets, pourraient dorénavant reposer sur une modélisation linguistique permettant d'indiquer plus finement le niveau de préférence. Il est aussi possible d'effectuer un partitionnement flou des domaines des variables considérées. Si on se limite à la variable D qui représente le jour du vol par rapport à une date fixe de conférence à laquelle la personne doit assister :

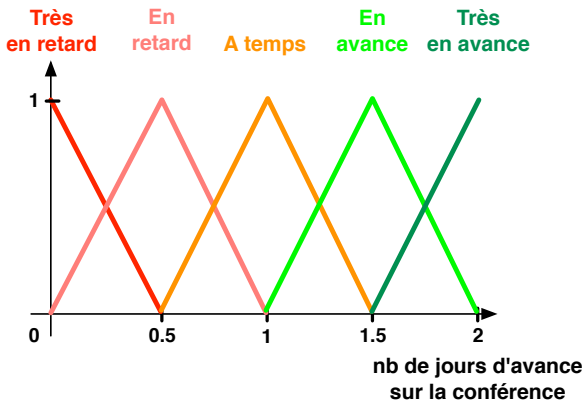

FIGURE 6.17 – Partitionnement flou d'un domaine temporel.

- Dans l'exemple de base, la variable D pouvait prendre la valeur D_{1d} (je prend l'avion seulement un jour avant la conférence) ou D_{2d} (je prend l'avion deux jours avant).
- Grâce aux SEF de l'approche linguistique floue mise en œuvre par les LCP-nets, on peut proposer le partitionnement suivant pour le domaine de la variable D (cf. figure 6.17) : *Très en retard*, *En retard*, *A temps*, *En avance*, *Très en avance*.

On utilise alors les nouveaux termes linguistiques sur D avec ceux du domaine *utilité* utilisés dans les exemples précédents de ce chapitre pour obtenir le LCP-net basique suivant (cf. figure 6.18, mis en comparaison avec la représentation CP-net de base, non linguistique, de la préférence. On

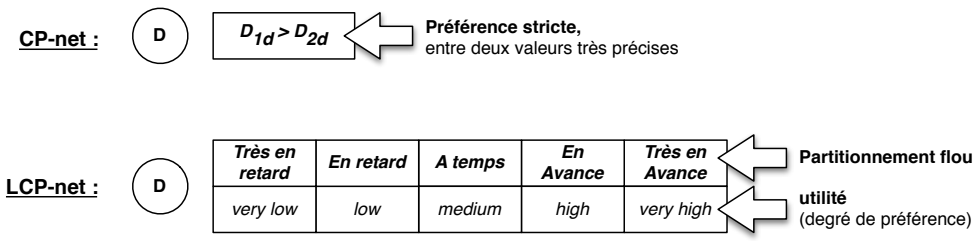

FIGURE 6.18 – CP-net *vs.* LCP-net : Préférence sur une date de départ en voyage.

constate que la préférence sous forme LCP-net est plus détaillée, tout en restant facilement intelligible pour l'utilisateur car entièrement linguistique. De plus, par le partitionnement flou, le modèle de préférences sera capable d'appréhender l'ensemble des cas intermédiaires entre 0 et 2 jours d'avance.

Pour finir, la prochaine étape serait de concrétiser les améliorations évoquées dans la section 6.5 en termes de traitement linguistique "de bout en bout" des modèles de préférences LCP-nets.

Chapitre 7

Composition agile de services

Sommaire

7.1 Introduction

A l'issue des présentations détaillées, mais séparées, de la composition *active*, puis *utile* de services, le but de ce chapitre est d'aborder l'intégration de ces deux contributions au sein d'une approche unifiée, déployée pour l'orchestration de services lors de l'exécution d'un processus métier défini dans le langage BPEL. Cette dernière contribution porte alors le nom de *composition agile de services*, qui traduit fidèlement la capacité d'une application répartie construite sur ses bases à s'adapter dynamiquement aux changements dans son environnement, par exemple en étant capable de prendre en considération l'évolution constante des services qui l'entourent (disparitions, déconnexions, variations dans la QoS, etc.), tout en tenant compte des contraintes non-fonctionnelles qui s'imposent à elle ainsi que des préférences des utilisateurs, exprimées à l'aide de LCP-nets.

Cette capacité, nous l'avons vue au chapitre 4 au cours du cas d'utilisation sur lequel s'appuient nos travaux, peut s'avérer déterminante lors de situations de crise, en particulier lors de la gestion

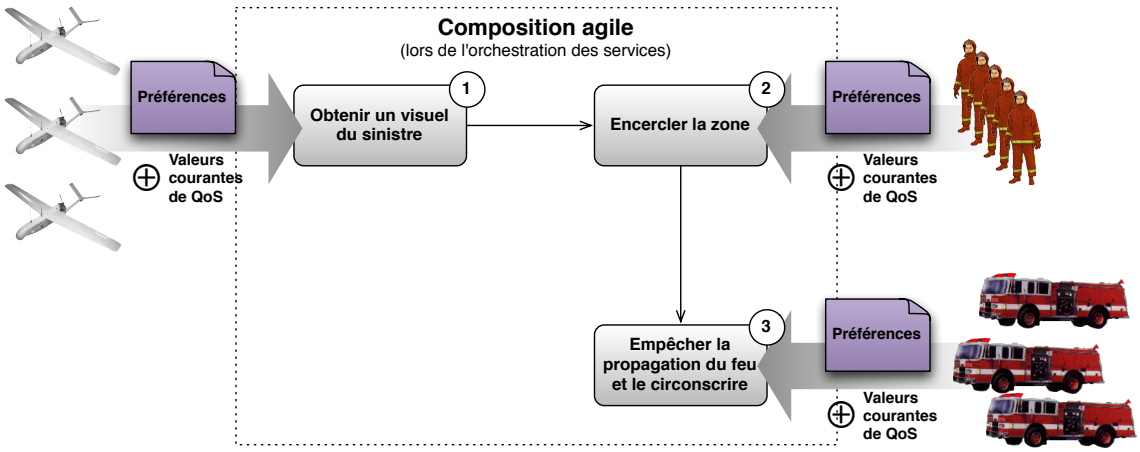

FIGURE 7.1 – Composition agile de service lors de la gestion d'un incendie.

d'incendies. Elle va alors permettre de choisir les meilleurs éléments disponibles du point de vue de leur Qualité de Service (parmi les drones, personnels et camions prêts à se rendre sur le lieu du sinistre) au moment même du déclenchement de la crise et pendant tout son déroulement, tout en permettant des ajustements "de dernière minute" par le biais des préférences utilisateur rattachées aux processus métiers (cf. figure 7.1). Ces préférences (contextuelles par nature) vont alors diriger la sélection des services, de manière active pendant tout le déroulement du processus de gestion de crise, sur la base des différents critères de QoS connus que l'on peut superviser. Par exemple : *"On souhaite, dans l'absolu, utiliser les drones disposant, pour la transmission des images, des meilleures valeurs en termes de sécurité, bande passante et résolution."*, ou encore *"On effectue un compromis entre l'optimisation de la bande passante et celle de la résolution"*.

Le déroulement de ce chapitre est le suivant : dans un premier temps, à la section 7.2, nous traiterons les considérations propres à l'intégration conjointe de la composition active et de la composition utile lors de l'orchestration de services. Au cours de cette section nous aborderons notamment le gain lié à la possibilité de factoriser les préférences LCP-nets dans les processus métiers. Dans un second temps, en section 7.3, nous détaillerons la syntaxe et la sémantique opérationnelle structurelle d'un langage de LCP-nets de plus haut niveau, à même de favoriser leur intégration au sein des processus métiers BPEL au format XML, sujet que nous aborderons à la section 7.4. Finalement, la dernière section 7.5 sera l'occasion d'évoquer la possibilité plus prospective d'intégrer la QoS globale des processus dans la liaison tardive des services.

7.2 Intégration conjointe des compositions actives et utiles dans l'orchestration de services

Le mécanisme de composition active de services, présenté au chapitre 5, a introduit de manière abstraite le concept primordial de *liaison tardive de services* ainsi que les différentes étapes qui précèdent son déclenchement. Celui de composition utile, au chapitre 6, est lié au *formalisme LCP-net* de préférences utilisateur. La finalité de cette section est alors d'aborder les problématiques plus

```
<?xml version="1.0" encoding="UTF-8"?>
<process

        <import importType="http://schemas.xmlsoap.org/wsdl/"
                location="../../test_bucket/service_libraries/tptp_EnginePrinterPort.wsdl"
                namespace="http://www.eclipse.org/tptp/choreography/2004/engine/Print" />

        //le partnerLinkType est defini dans le WSDL importe par le processus !
        <partnerLinks>
                <partnerLink    name="printService"
                                partnerLinkType="print:printLink"
                                partnerRole="printService"/>
        </partnerLinks>

        (...)

        //l'invocation passe forcement par une abstraction partnerLink
        <invoke partnerLink="printService" operation="print" inputVariable="hello_world" />

</process>
```

Code 7.1 – Extrait d'un processus métier basique au format BPEL.

concrètes de leur intégration *conjointe*, dans un cadre non-fonctionnel, au sein de l'orchestration de services issus de l'exécution d'un processus métier au format BPEL.

7.2.1 Définition d'une activité BPEL de liaison et invocation tardive de service

D'un point de vue technique, dans une architecture SOA, la définition d'un processus métier effectuée en suivant la spécification WS-BPEL 2.0 de base (c'est-à-dire sans exploiter le mécanisme d'extension du langage) ne permet pas la mise en place d'une liaison tardive de services : sa syntaxe implique qu'un et un seul service soit explicitement affecté à chaque site d'invocation de service lors de la définition du processus. Cette contrainte est illustrée par l'extrait de code BPEL 7.1, déjà rencontré, qui cherche à effectuer l'affichage de la chaîne de caractères *"Hello World !"* en faisant appel à un service Web d'affichage dédié. Ce service doit être connu et indiqué au moment de la définition du processus car l'instruction d'invocation de service requière un unique *partnerLink* nommé *'printService'* qui est lié à une structure *portType* d'un fichier WSDL *'...tptp_EnginePrinterPort.wsdl'* importé auparavant et dont on reproduit ici un extrait (cf. Extrait 7.2). Or, chaque *portType* correspond à la définition d'un service Web spécifique, accessible au travers d'un *port* qui spécifie l'adresse concrète du service. Il n'est donc pas possible de choisir un service à un autre moment que celui de la définition, ni de spécifier une liste d'alternatives pour l'exécution. Cette remarque est valable *ceteris paribus*, que l'offre de service Web soit au format WSDL 1.1 ou 2.0.

A contrario, dans notre cadre SSOA, au cours de l'orchestration des services effectuée à partir de processus métiers abstraits, les calculs modélisés dans les processus doivent se retrouver concrétisés sous la forme de flots de contrôle structurés où les besoins abstraits sont transformés en appels aux services effectivement disponibles dans l'environnement d'exécution des processus, et pas uniquement dans celui de leur définition. Cependant, si la spécification WS-BPEL 2.0 prévoit effectivement un mécanisme d'extension des opérations de base du langage, voire de définition d'opérations totalement nouvelles, aucune extension relative à un mécanisme d'invocation tardive des services n'est prédéfinie.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/" (...) >

    <portType name="Print">
        (..)
    </portType>

        //lie au portType 'Print'
    <binding name="PrintPortWsifBinding" type="tns:Print">
        (..)
    </binding>

        //lie au binding 'PrintPortWsifBinding'
    <service>
        <port name="JavaPrintPort" binding="tns:PrintPortWsifBinding">
            //adresse concrete d'acces au service d'affichage
        </port>
    </service>

        //lie au portType 'Print'
    <partnerLinkType name="printLink">
        <role name="printService" portType="tns:Print"/>
    </partnerLinkType>

</definitions>
```

Code 7.2 – Extrait d'une offre basique de service Web d'affichage.

Telles que nous les avons déjà abordées (cf. section 5.3), les étapes successives de *filtrage des services*, d'*initialisation de leur supervision* et finalement de *liaison tardive* sont indispensables à la pleine réalisation de la composition active des services. Le choix de la répartition temporelle des deux premières étapes par rapport au cycle de vie des processus métier découle d'une logique d'optimisation du temps de réponse de ces derniers. A titre d'exemple, le calcul des modèles d'adaptation de données ne pourrait être réalisé de manière pleinement satisfaisante s'il était effectué au cours de l'orchestration des services : l'impact de ses algorithmes d'appariement étant alors beaucoup trop important par rapport aux caractéristiques d'agilité que l'on est en droit d'attendre de systèmes répartis, en particulier déployés pour de la gestion de crise.

La répartition que nous établissons est alors illustrée par la figure 7.2 : les étapes de filtrage et d'initialisation de la supervision étant alors effectuées au chargement du processus, cela implique que la liste des services valides filtrés (les pompiers sur cet exemple) et les valeurs de QoS courantes de ces services (résistance au feu, fatigue physique, fréquence radio évoluant à l'intérieur de bornes contractuelles), obtenues au travers des sondes du canevas de supervision, seront disponibles lors de la liaison tardive des services pendant l'exécution du processus. Cette répartition signifie aussi que seule la liaison tardive des services, et leur invocation subséquente, reste effectivement à traiter au cours de l'orchestration. Dans l'implantation de référence du projet SemEUsE, filtrage et supervision sont effectivement pris en charge par des composants externes à l'orchestration et la liaison tardive des services : par un mécanisme de composition dynamique proche des travaux que nous avons étudiés dans l'état de l'art pour le premier, et par le canevas mis au point par Le Duc *et al.* pour le second [Le Duc et al., 2009].

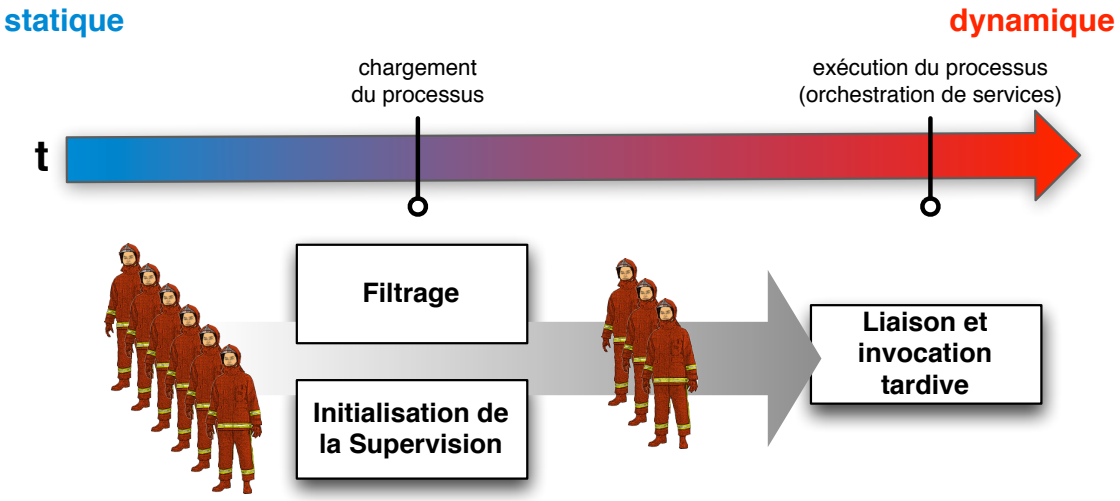

FIGURE 7.2 – Répartition temporelle des étapes clés de la coordination active de services.

Nous définissons alors une "activité étendue" BPEL dont les détails d'implantation seront abordés au chapitre 9. La syntaxe de cette activité, nommée *lateBindingInvoke*, est donnée par le fragment de code 7.3 au format XML :

```
<lateBindingInvoke inputVariable="invoke_in" outputVariable="invoke_out" preference="
    preferences.lcpnet">
        <candidateServices>
                <service EPR="service1" portType="service1PortType" operation="getVideo"
                    contract="negociated_wsag_1.xml"/>
                <service EPR="service2" portType="service2PortType" operation="getVideo"
                    contract="negociated_wsag_2.xml"/>
                (...)
        </candidateServices>
</lateBindingInvoke>
```

Code 7.3 – Activité BPEL *lateBingingInvoke* au format XML

L'activité de liaison et d'invocation tardive de service dispose ainsi :

- de deux arguments *inputVariable* et *outputVariable* qui pointent vers les variables BPEL utilisées pour le stockage des paramètres et de la valeur de retour du service qui sera appelé lors de l'exécution de cette activité BPEL,
- d'un argument *preference* qui permet d'indiquer un fichier de préférence au format LCP-net à utiliser lors de la sélection du service à invoquer,
- d'une *liste de services valides* fonctionnellement ("ce sont bien des pompiers") et non-fonctionnellement ("ils correspondent bien aux critères généraux de QoS exprimés sous forme de bornes dans les contrats") parmi lesquels le service à invoquer sera déterminé au dernier moment (en liaison tardive) cette liste a été déterminée précédemment par l'étape de filtrage. Pour chaque service, on dispose des mêmes informations que dans un BPEL classique (réfé-

rence au point d'accès *EPR*, interface distante *portType* et opération à appeler), ainsi que de son contrat de Qualité de Service.

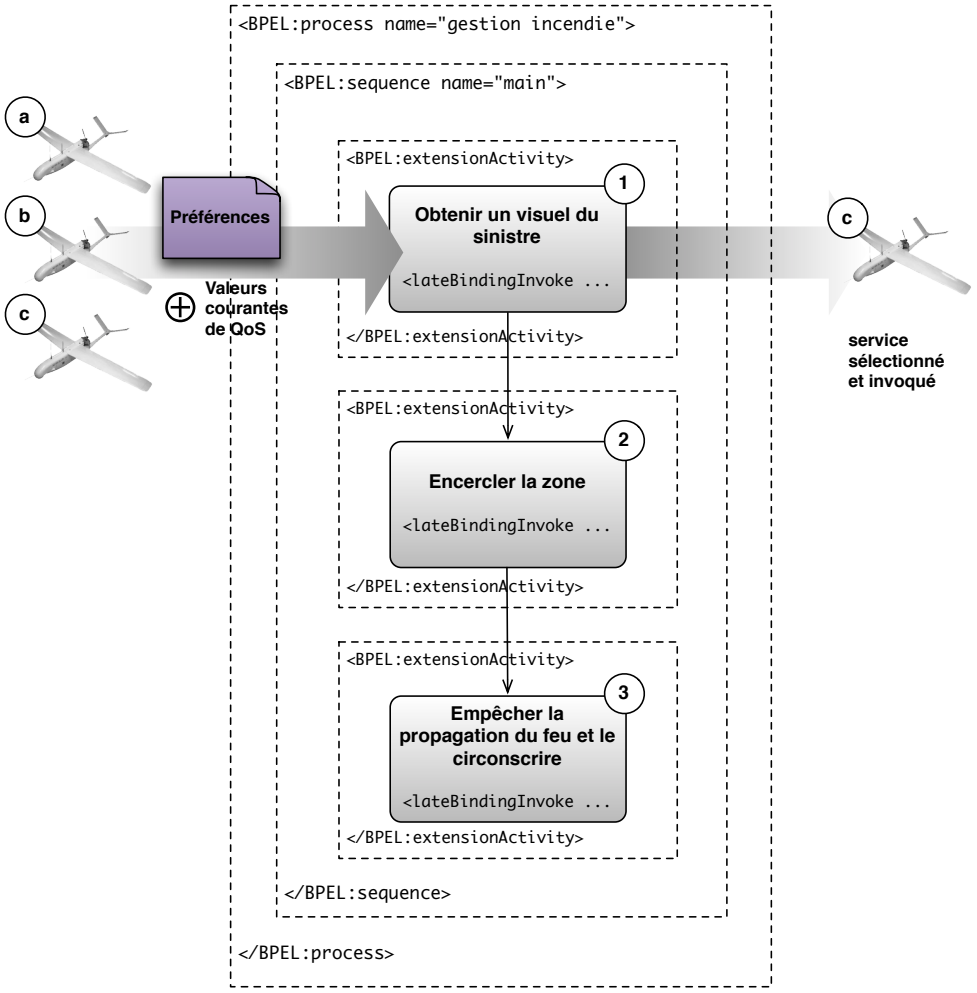

FIGURE 7.3 – Activité *lateBindingInvoke* dans le processus BPEL de gestion de incendie.

Sur la base de notre précédent exemple pour la gestion de crise, c'est cette instruction qui va être utilisée dans le processus métier BPEL au niveau des sites d'appels n°1, 2 et 3. Sans entrer dans les détails techniques, la figure 7.3 illustre l'usage qui en est fait, tout en précisant la nature du processus BPEL : les trois activités *lateBindingInvoke*, identifiées explicitement dans le processus comme activités étendues, sont incluses dans une activité de base du type *sequence*, elle même comprise dans un élément *process*.

7.2.2 Définition incrémentale des préférences utilisateur

Est apparu au cours de l'intégration conjointe de la liaison tardive et des LCP-nets, tel que présenté ci-dessus, l'intérêt manifeste de permettre la factorisation des modèles, de manière à éviter d'inutiles répétitions dans les préférences utilisateur. On rationaliserait ainsi l'utilisation des

préférences en réduisant les répétitions, la conséquence directe étant une *réduction des coûts de développement* en favorisant le principe de *réutilisabilité* mis en avant par l'architecture SOA.

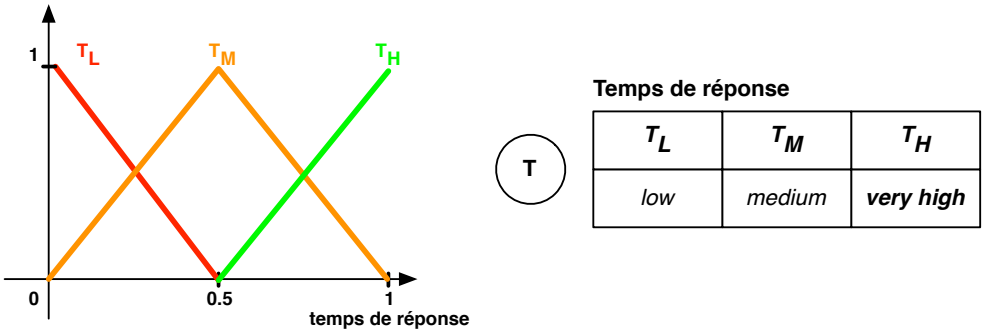

FIGURE 7.4 – LCP-net modélisant la préférence sur le temps de réponse de tous les services.

En effet, lors de la définition de processus métiers, les programmeurs peuvent avoir des préférences générales qui s'appliquent à plusieurs sites d'appels de services (c'est-à-dire à plusieurs *lateBindingInvoke*), avec éventuellement quelques modifications à leur apporter pour les adapter aux spécificités de chaque site. Par rapport à notre cas d'utilisation, il pourrait par exemple s'agir d'une propriété sur le temps de réponse des services, effectivement partagée par les drones, personnels et camions, et pour laquelle notre préférence est toujours de la forme *"Je souhaite obtenir le temps de réponse le plus faible possible"*, quel que soit le type de service. Cette préférence correspond au LCP-net de la figure 7.4, ici accompagné du partitionnement flou du domaine du *temps de réponse* en *ms*, préalablement normalisé sur $[0, 1]$ qui donne les différentes fonctions d'appartenance des termes linguistiques T_L, T_M et T_H (respectivement *temps de réponse faible*, *moyen* et *élevé*).

Tel que défini actuellement, le langage des LCP-nets conduirait à la définition de trois modèles distincts de préférence (sur la base de ceux introduits dans la section 6.3.2) au sein desquels la préférence globale sur le temps de réponse serait systématiquement reportée, telle que représentée par la figure 7.5. On cherche alors à changer cet état de fait par un mécanisme de définition incrémentale (ou "différentielle") des modèles de préférences : les *fragments*.

7.2.3 Notion de fragment de préférences

Plutôt que de devoir répéter systématiquement, dans le LCP-net local à chaque site d'appel, une préférence applicable à un ensemble de sites (dans notre exemple, la préférence sur le temps de réponse des services), une alternative plausible serait de factoriser ces préférences communes dans un *LCP-net partagé*, puis d'ajouter sur chaque site d'appel les variations locales à lui appliquer pour obtenir le *LCP-net effectif* à utiliser lors de la liaison tardive de services.

Cette capacité à définir les préférences utilisateur de manière incrémentale, par des extensions ou modifications locales, a été ajoutée au formalisme LCP-nets par l'introduction de la notion de *fragment* de préférences. Les fragments sont des entités, construites sur la base d'un LCP-net existant, qui décrivent de manière non ambiguë des *ajouts*, *suppressions*, ou *modifications* de tout élément sur leur LCP-net de base.

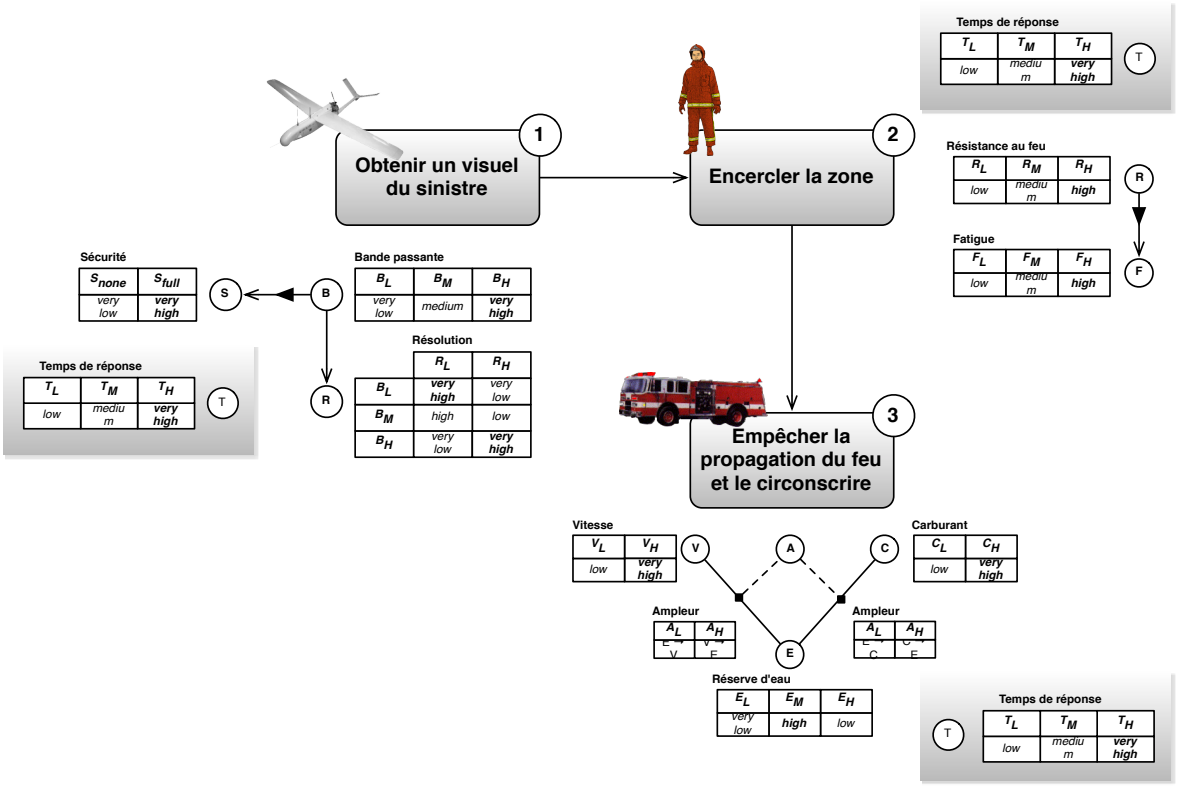

FIGURE 7.5 – Dilution d'une préférence globale dans un processus métier.

Pour que sa définition reste la plus simple possible, la forme générale d'un fragment suit le même modèle et la même représentation graphique qu'un LCP-net classique, tel qu'on peut le voir sur la figure 7.7 :

- Pour chaque construction de base du formalisme LCP-net (le LCP-net lui-même, ses nœuds, arcs et tables), il existe un type de fragment dérivé : on dispose donc de fragments de LCP-nets, de nœuds, d'arcs et de tables. Sur la figure on distingue un fragment de LCP-net contenant lui-même deux fragments de nœud et un fragment d'arc.

- Pour chaque fragment il faut indiquer l'*élément cible*, situé dans un modèle LCP-net de base, auquel il s'applique. Dans notre exemple, ces liens ne sont pas retranscrits sur la figure, cependant, le fragment de LCP-net s'applique bien à notre précédent LCP-net d'imagerie, de même que les fragments de nœuds à leurs nœuds respectifs dans ce modèle cible, etc.

- Un fragment peut servir à modifier les propriétés de son élément de base ou à le supprimer (on parle alors d'anti-fragment). Dans notre exemple, on dispose d'un anti-fragment d'arc (de B vers R) et d'un anti-fragment de nœud (pour R) ; ce qui signifie que ces deux éléments cibles seront supprimés lors de l'application du LCP-net.

- Lors de la modification des propriétés d'un élément de base par un fragment, il est possible de décrire des ajouts. Ces ajouts sont exprimés sous la forme d'éléments d'un type de base du formalisme LCP-net, c'est-à-dire que pour ajouter un nœud à un LCP-net il faut créer en

premier lieu un fragment de LCP-net, et ajouter un *véritable* nœud à sa liste vierge de nœuds. C'est le cas sur la figure, où pour ajouter un nouveau nœud Z on crée un élément de type *'nœud'*, et non de type *'fragment de nœud'*.

Cette nouvelle fonctionnalité de factorisation va effectivement permettre la définition de préférences globales qui pourront être utilisées comme références de base dans un processus métier, suivies de leur raffinement par l'utilisation d'un fragment de préférence sur chaque site d'appel de service nécessitant des ajustements spécifiques : comme le fait d'accorder une plus grande importance à une dimension de QoS particulière ou, au contraire, d'en ignorer certaines.

S'ajoute alors aux bénéfices en termes de rationalisation des coûts de développement, un autre avantage, particulièrement pertinent dans notre contexte SSOA, qu'est celui de l'*amélioration de la lisibilité* et de la compréhension des préférences, par la facilitation de la reconnaissance des préférences communes aux sites d'appels.

Première illustration du mécanisme de définition incrémentale

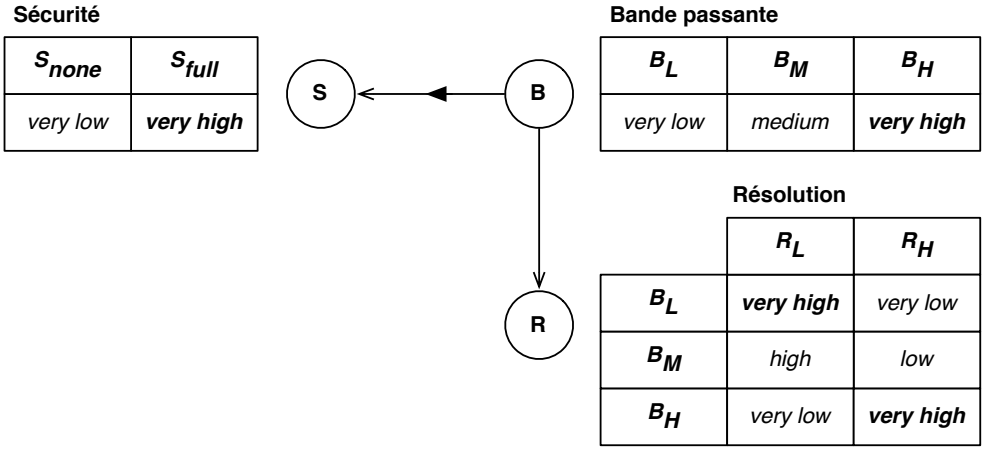

FIGURE 7.6 – Préférence *ImagingServicePreference* sous forme graphique.

Pour illustrer ce mécanisme de définition incrémentale de manière plus exhaustive, on compose tout d'abord le LCP-net *ImagingServicePreference* de préférence sur la sélection d'un drone d'imagerie aérienne qui avait déjà été introduit au chapitre 6 (cf. figure 7.6), avec le fragment *ImagingServiceFragment* reproduit sous forme graphique par la figure 7.7, de manière à obtenir un nouveau LCP-net $ImagingServicePreference_{new}$ tel que :

- la préférence sur la résolution R est remplacée par une préférence sur une nouvelle propriété de zoom Z,
- un nouveau lien de type *i-arc* est introduit entre B et Z,
- les valeurs d'utilité sur la CPT du nœud B sont changées.

Un extrait de la sérialisation au format XML de ce fragment *ImagingServiceFragment* est donné par le code 7.4, il met en exergue :

```
<?xml version="1.0" encoding="UTF-8"?>
<fragments:LCPnetFragment xmi:version="2.0" (...) fragmentName="ImagingServiceFragment">
  <nodes xsi:type="fragments:LNodeFragment" name="" fragmentName="R_antifragment" antiFragment="true">
    <baseLNode href="imaging_service_preference.lcpnet#//@nodes.2"/>
  </nodes>
  <nodes xsi:type="LCPnet:LNode" name="Z" inArcs="//@arcs.1" valueDomain="//@valueDomains.0">
    <domain xsi:type="LCPnet:LNodeValue" name="Zoom_Low" linguisticValue="//@valueDomains.0/@subsets.0"/>
    <domain xsi:type="LCPnet:LNodeValue" name="Zoom_Medium" linguisticValue="//@valueDomains.0/@subsets.1"/>
    <domain xsi:type="LCPnet:LNodeValue" name="Zoom_High" linguisticValue="//@valueDomains.0/@subsets.2"/>
    <linguisticTable name="zoom_clpt">
      <lines>
        <utility nodeValue="//@nodes.1/@domain.0">
          <utility href="imaging_service_preference.lcpnet#//@utilityDomain/@subsets.1"/>
        </utility>
        <utility nodeValue="//@nodes.1/@domain.1">
          <utility href="imaging_service_preference.lcpnet#//@utilityDomain/@subsets.2"/>
        </utility>
        <utility nodeValue="//@nodes.1/@domain.2">
          <utility href="imaging_service_preference.lcpnet#//@utilityDomain/@subsets.3"/>
        </utility>
      </lines>
    </linguisticTable>
  </nodes>
  <nodes xsi:type="fragments:LNodeFragment" name="" outArcs="//@arcs.1" fragmentName="B_Modifier">
    <linguisticTable xsi:type="fragments:CLPTFragment" name="">
      <lines xsi:type="fragments:CLPTLineFragment">
        <utility xsi:type="fragments:LNodeValueUtilityFragment" name="">
          <utility href="imaging_service_preference.lcpnet#//@utilityDomain/@subsets.1"/>
          <baseLNodeValueUtility href="imaging_service_preference.lcpnet#//@nodes.1/@linguisticTable/@lines.0/@utility.0"/>
        </utility>
        <utility xsi:type="fragments:LNodeValueUtilityFragment" name="">
          <utility href="imaging_service_preference.lcpnet#//@utilityDomain/@subsets.3"/>
          <baseLNodeValueUtility href="imaging_service_preference.lcpnet#//@nodes.1/@linguisticTable/@lines.0/@utility.2"/>
        </utility>
        <baseCLPTLine href="imaging_service_preference.lcpnet#//@nodes.1/@linguisticTable/@lines.0"/>
      </lines>
      <baseCLPT href="imaging_service_preference.lcpnet#//@nodes.1/@linguisticTable"/>
    </linguisticTable>
    <baseLNode href="imaging_service_preference.lcpnet#//@nodes.1"/>
  </nodes>
  <arcs xsi:type="fragments:ArcFragment" name="" fragmentName="BtoR_antifragment" antiFragment="true">
    <baseArc href="imaging_service_preference.lcpnet#//@arcs.1"/>
  </arcs>
  <arcs xsi:type="LCPnet:IArc" name="BtoZ" startNode="//@nodes.2" endNode="//@nodes.1"/>
  <valueDomains name="Zoom">
    <subsets name="Zoom_Low">
      <fuzzySubset y="1.0"/>
      <fuzzySubset x="0.5"/>
    </subsets>
    <subsets name="Zoom_Medium">
      <fuzzySubset/>
      <fuzzySubset x="0.5" y="1.0"/>
      <fuzzySubset x="1.0"/>
    </subsets>
    <subsets name="Zoom_High">
      <fuzzySubset x="0.5"/>
      <fuzzySubset x="1.0" y="1.0"/>
    </subsets>
  </valueDomains>
  <baseLCPnet href="imaging_service_preference.lcpnet#/"/>
</fragments:LCPnetFragment>
```

Code 7.4 – Fragment *ImagingServiceFragment* sérialisé sous forme XML.

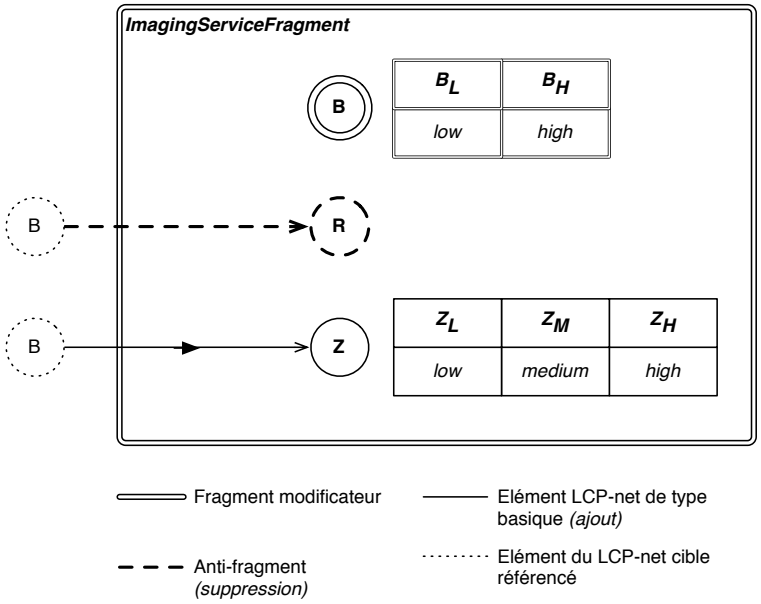

FIGURE 7.7 – Fragment *ImagingServiceFragment* sous forme graphique.

– l'import initial du modèle LCP-net de base *imaging_service_preference.lcpnet* ;

– la suppression du nœud R et de sa CPT associée, par la définition d'un anti-fragment *R_antifragment* ;

– l'introduction d'un nouveau nœud Z et de l'arc *BtoZ* ;

– l'utilisation d'expressions XPath pour désigner tout élément défini au sein du fichier *imaging_service_preference.lcpnet*.

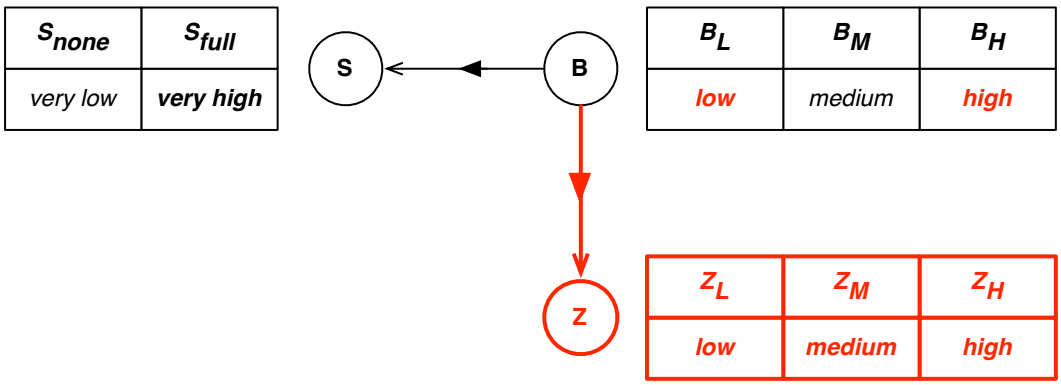

FIGURE 7.8 – LCP-net $ImagingServicePreference_{new}$ résultant de l'application de *ImagingServiceFragment*.

Le nouveau LCP-net $ImagingServicePreference_{new}$, totalement indépendant, résultant de l'application du fragment *ImagingServiceFragment*, est représenté de manière graphique sur la figure 7.8. Les éléments ajoutés ou modifiés sont indiqués en rouge.

Seconde illustration du mécanisme de définition incrémentale

On cherche maintenant à factoriser la préférence globale sur le temps de réponse dans le processus BPEL représenté par la figure 7.5. Pour ce faire :

- on factorise la partie du modèle de préférences concernant le temps de réponse T dans un LCP-net unique,
- les préférences locales à chaque site d'appels *sont reformulées sous la forme de fragments*,
- chaque fragment local va importer le LCP-net unique sur le temps de réponse et y ajouter ses propres spécificités.

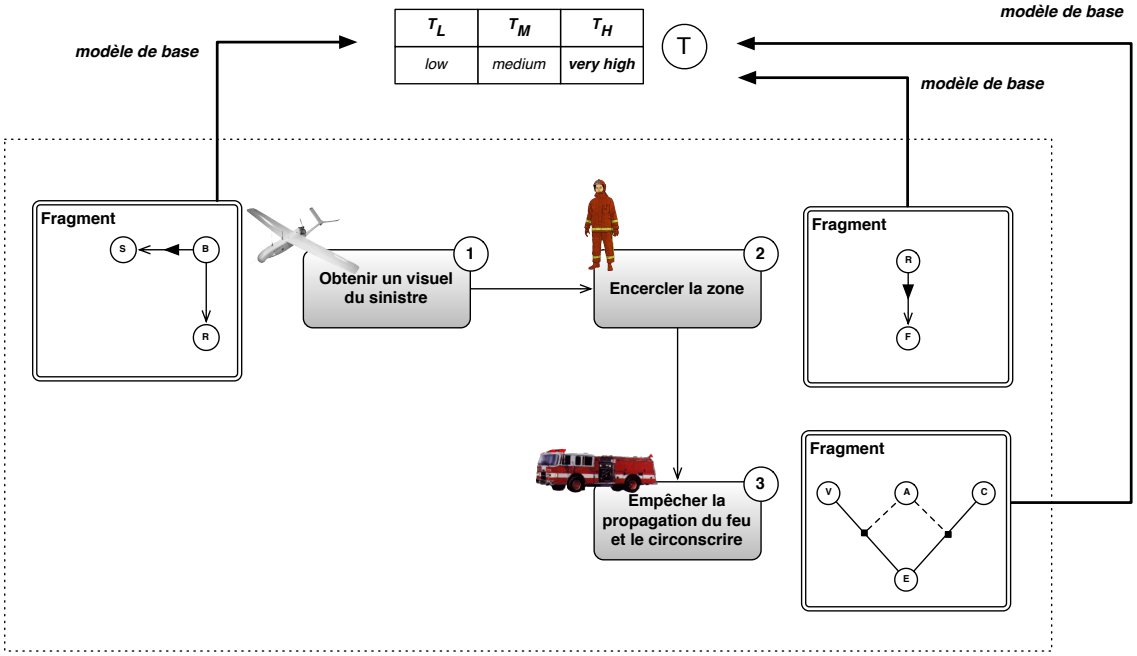

FIGURE 7.9 – Factorisation d'un préférence globale à un processus métier.

Cette approche est illustrée par la figure 7.9 où les tables des préférences locales ont été volontairement omises de manière à ne pas nuire à sa lisibilité. Lors de l'évaluation de chaque site d'appel de services, la préférence effective considérée sera celle obtenue par application du fragment local à la préférence globale sur T.

7.3 Représentation de plus haut niveau des LCP-nets

Dans le cadre de la composition agile de services, on cherche à intégrer les préférences utilisateur au format LCP-net ainsi que leurs fragments au sein des orchestrations de services. Pour que cette intégration soit couronnée de succès, il faut minimiser les contraintes que devra subir un développeur de processus métier BPEL pour les y intégrer.

Dans l'implantation actuelle du langage LCP-net, ainsi que celle du moteur de raisonnement sur préférences, la représentation XML des modèles correspond en fait à celle de leur sérialisation

automatique, fondée sur le canevas EMF. Des exemples de cette sérialisation ont été précédemment donnés par les fragments de code 6.1 et 7.4.

Bien que lisibles en l'état, ces représentations n'en restent pas moins difficiles d'accès pour le développeur de processus et ne pourraient être intégrées directement dans les processus BPEL, car cela signifierait que des modèles de préférences externes au processus, *spécifiés à un niveau de langage inférieur*, devraient y être importés. Cette implantation, *à elle seule*, ne permet donc pas une intégration convenable des LCP-nets et de leurs fragments dans les processus BPEL.

La solution que nous avançons pour résoudre cette problématique consiste à mettre au point une représentation XML "haut niveau" des LCP-nets et fragments, de manière à homogénéiser le niveau d'abstraction du langage permettant d'exprimer les préférences avec celui de BPEL. On désigne alors les modèles de préférences et fragments décrits dans ce langage par *HL*-LCP-nets ("High Level LCP-nets") et *HL*-LCP-frags ("High Level LCP-nets fragments") respectivement. En conséquence de quoi, dans la suite de ce document, le terme "LCP-net" seul servira à designer en fonction du contexte, le concept abstrait de préférence utilisateur dans ce formalisme, ou bien les LCP-nets "bas niveau" telles qu'implantés par le canevas de modélisation et de décision sur LCP-nets (cf. section 9.2).

Cette syntaxe permet de se rendre indépendant du canevas sous-jacent de modélisation et décision sur préférences utilisateur, présenté au chapitre 9 comme une machine virtuelle sur les LCP-nets de bas niveau, et dont l'ensemble des primitives est consultable à la section 9.2.2. Nous exploitons alors ces primitives pour définir la sémantique opérationnelle structurelle des *HL*-LCP-nets et *HL*-LCP-frags (cf. section 7.3.3) *via* un ensemble de règles de production qui établissent les correspondances entre leur représentation abstraite (cf. section 7.3.2) et les primitives de cette machine virtuelle à appeler pour construire et exploiter les LCP-nets bas niveau correspondants.

7.3.1 Syntaxe XML

La mise au point d'une syntaxe XML de plus haut niveau pour les LCP-nets et leurs fragments va donc rendre possible leur intégration au cœur même des processus métier BPEL. On l'illustre par les deux exemples suivants.

La représentation XML du *HL*-LCP-net correspondant au réseau de préférences nommé *ImagingServicePreference*, déjà rencontré, sur la sélection d'un service d'imagerie, est donnée ci-après par l'extrait de code 7.5. Elle est structurée, comme pour tout autre *HL*-LCP-net de la façon suivante :

- les premiers éléments *utilityDomain* et *valueDomains* du réseau de préférences donnent respectivement les partitionnements flous du domaine d'utilité des services utilisés dans les tables, et ceux de tous les domaines de valeurs de QoS considérés (ici la sécurité de la communication *Security*, la bande passante *Bandwidth* et la résolution d'image *Resolution*),
- sous l'élément *nodes* est défini l'ensemble des nœuds (*node*) du réseau auxquels sont attachées les tables de préférences conditionnelles (*table*). La correspondance entre la forme graphique du nœud R et sa syntaxe *HL*-LCP-net est illustrée par la figure 7.10,
- la définition des nœuds est suivie de celle des arcs, sous l'élément *arcs*. Dans cet exemple figurent deux des types d'arcs disponibles : un *arc* (élément "BtoR") et un *i-arc* (élément

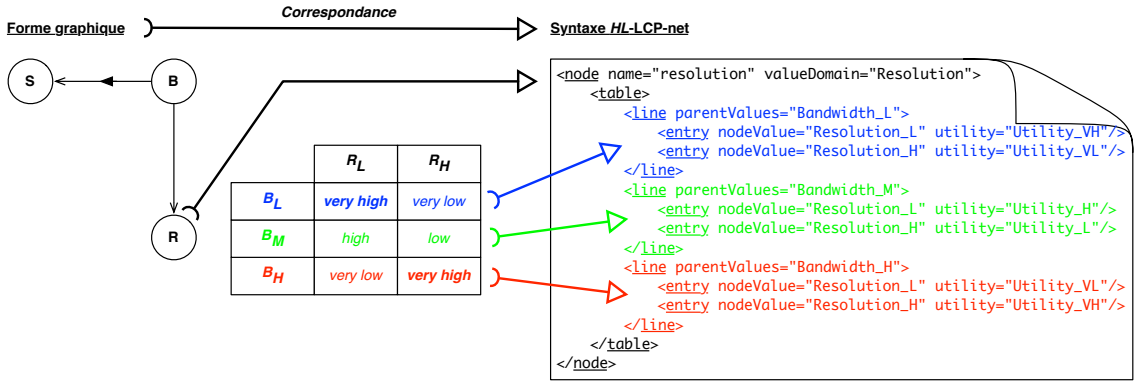

FIGURE 7.10 – Correspondance entre la forme graphique du nœud R et sa syntaxe *HL*-LCP-net.

"BtoS"). Pour illustrer le type *ci-arc* qui n'est pas instancié dans ce modèle, on se reportera à l'extrait de *HL*-LCP-net donné par le code 7.6.

La syntaxe XML de haut-niveau pour un fragment de préférence *ImagingServiceFragment* est quant à elle illustrée par l'extrait de code 7.7. Elle suit la même forme générale que celle d'un *HL*-LCP-net, il faut cependant noter que :

- la distinction entre les éléments de base de type *HL*-LCP-net et ceux de fragments est effectuée sur leur nom : à chaque type de base x correspond un type fragment *xFragment*,
- tout comme expliqué au chapitre 6, des éléments de type *HL*-LCP-net peuvent apparaître au sein des fragments, ils servent à ajouter des nouveaux éléments dans le LCP-net effectif résultant de l'application du fragment : dans cet exemple, on ajoute un nouveau domaine de valeurs (type *valueDomain*) appelé *Zoom*, un nouveau nœud *zoom* avec sa table, ainsi qu'un nouvel i-arc de *bandwidth* à *zoom*,
- pour chaque élément de type fragment où l'information est pertinente, le paramètre booléen *antiFragment* permet d'indiquer s'il s'agit d'un anti-fragment de suppression (paramètre à *true*), ou d'un simple fragment de modification (paramètre à *false*),
- pour chaque élément de type fragment où l'information est nécessaire, il faut indiquer la base (paramètre *base*) du fragment, c'est-à-dire l'élément du LCP-net de base qu'il modifie ou supprime.

7.3.2 Syntaxe abstraite de ce nouveau langage

On définit, par la figure 7.11, la grammaire d'arbres d'une syntaxe abstraite équivalente à celle XML utilisée précédemment pour représenter les *HL*-LCP-nets, mais moins verbeuse, de manière à en augmenter la lisibilité et faciliter son intégration dans les règles de production de la section 7.3.3. Il s'agit, en définitive, de la syntaxe d'un nouveau langage pour la représentation des préférences LCP-nets.

Le même principe est appliqué aux fragments. La grammaire présentée dans la figure 7.12 couvre l'ensemble des expressions *HL*-LCP-frag.

```
<lcpnet name="Security_Camera_Preference">
        <utilityDomain name="Utility" lowerBound=0.0 upperBound=1.0>
                <subset name="Utility_VL">
                        <coordinate x="0.0" y="1.0" />
                        <coordinate x="0.25" y="0.0" />
                </subset>
                <subset name="Utility_L">
                        <coordinate x="0.0" y="0.0" />
                        <coordinate x="0.25" y="1.0" />
                        <coordinate x="0.5" y="0.0" />
                </subset>
                (...)
        </utilityDomain>

        <valueDomains>
                <valueDomain name="Security" lowerBound="0.0" upperBound="1.0">
                        (...)
                </valueDomain>
                <valueDomain name="Bandwidth" lowerBound="0.0" upperBound="1.0">
                        (...)
                </valueDomain>
                <valueDomain name="Resolution" lowerBound="0.0" upperBound="1.0">
                        (...)
                </valueDomain>
        </valueDomains>

        <nodes>
                <node name="security" valueDomain="Security">
                        <table>
                                <line>
                                        <entry nodeValue="Security_None" utility="Utility_VL"/>
                                        <entry nodeValue="Security_Full" utility="Utility_VH"/>
                                </line>
                        </table>
                </node>

                <node name="bandwidth" valueDomain="Bandwidth">
                        <table>
                                <line>
                                        <entry nodeValue="Bandwidth_L" utility="Utility_VL"/>
                                        <entry nodeValue="Bandwidth_M" utility="Utility_M"/>
                                        <entry nodeValue="Bandwidth_H" utility="Utility_VH"/>
                                </line>
                        </table>
                </node>

                <node name="resolution" valueDomain="Resolution">
                        <table>
                                <line parentValues="Bandwidth_L">
                                        <entry nodeValue="Resolution_L" utility="Utility_VH"/>
                                        <entry nodeValue="Resolution_H" utility="Utility_VL"/>
                                </line>
                                <line parentValues="Bandwidth_M">
                                        <entry nodeValue="Resolution_L" utility="Utility_H"/>
                                        <entry nodeValue="Resolution_H" utility="Utility_L"/>
                                </line>
                                <line parentValues="Bandwidth_H">
                                        <entry nodeValue="Resolution_L" utility="Utility_VL"/>
                                        <entry nodeValue="Resolution_H" utility="Utility_VH"/>
                                </line>
                        </table>
                </node>
        </nodes>

        <arcs>
                <iarc name="BtoS" startNode="bandwidth" endNode="security"/>
                <arc name="BtoR" startNode="bandwidth" endNode="resolution"/>
        </arcs>
</lcpnet>
```

Code 7.5 – Syntaxe *HL*-LCP-net de *ImagingServicePreference*.

```
<arcs>
        <ciarc name="Y-Z">
                <table>
                        <line selectorSetValues="Bandwidth_L, Resolution_H" startNode="Y" endNode="Z"/>
                        <line selectorSetValues="Bandwidth_H, Resolution_L" startNode="Z" endNode="Y"/>
                </table>
        </ciarc>
</arcs>
```

Code 7.6 – Syntaxe *HL*-LCP-net d'un ci-arc.

```
<lcpnetFragment name="Security_Camera_Preference_Fragment" base="Security_Camera_Preference">
        <valueDomains>
                <valueDomain name="Zoom" lowerBound="0.0" upperBound="1.0">
                        (...)
                </valueDomain>
        </valueDomains>

        <nodes>
                <node name="zoom" valueDomain="Zoom">
                        <table>
                                <line>
                                        <entry nodeValue="Zoom_L" utility="Utility_L"/>
                                        <entry nodeValue="Zoom_M" utility="Utility_M"/>
                                        <entry nodeValue="Zoom_H" utility="Utility_H"/>
                                </line>
                        </table>
                </node>

                <nodeFragment name="resolution_antiFragment" antiFragment="true" base="resolution"/>

                <nodeFragment name="bandwidth_modifierFragment" antiFragment="false" base="bandwidth">
                        <tableFragment>
                                <lineFragment base="0">
                                        <entryFragment nodeValue="Bandwidth_L" utility="Utility_L"/>
                                        <entryFragment nodeValue="Bandwidth_H" utility="Utility_H"/>
                                </lineFragment>
                        </tableFragment>
                </nodeFragment>
        </nodes>

        <arcs>
                <iarc name="BtoZ" startNode="bandwidth" endNode="zoom"/>
                <arcFragment name="BtoR_antifragment" antiFragment="true" base="BtoR"/>
        </arcs>
</lcpnetFragment>
```

Code 7.7 – Syntaxe *HL*-LCP-frag de *ImagingServiceFragment*.

```
  l   ::=  lcpnet ud vd* n* a*
 ud   ::=  utilityDomain id lb ub s*
  s   ::=  subset id c*
  c   ::=  coordinate real real
 vd   ::=  valueDomain id lb ub s*
  n   ::=  node id id cpt
cpt   ::=  table cpli*
cpli  ::=  line id* e*
  e   ::=  entry id id
  a   ::=  ia | cia | na
 ia   ::=  iarc id id id
cia   ::=  ciarc id cit
 na   ::=  arc id id id
cit   ::=  table cil*
cil   ::=  line id* id id
 lb   ::=  real
 up   ::=  real
```

FIGURE 7.11 – Grammaire d'arbre des *HL*-LCP-nets.

$$
\begin{array}{rcll}
l_{frag} & ::= & & \underline{\text{lcpnetFragment}}\ [ud_{frag}]\ [vd_{frag}*]\ [n_{frag}*]\ [a_{frag}*] \\
ud_{frag} & ::= & ud & | \ \underline{\text{utilityDomainFragment}}\ \text{id isAntiFrag id [lb] [ub] } [s_{frag}*] \\
s_{frag} & ::= & s & | \ \underline{\text{subsetFragment}}\ \text{id isAntiFrag id [c*]} \\
vd_{frag} & ::= & vd & | \ \underline{\text{valueDomainFragment}}\ \text{id isAntiFrag id [lb] [ub] } [s_{frag}*] \\
n_{frag} & ::= & n & | \ \underline{\text{nodeFragment}}\ \text{id isAntiFrag id [id] } [cpt_{frag}] \\
cpt_{frag} & ::= & cpt & | \ \underline{\text{tableFragment}}\ \text{isAntiFrag } [cpli_{frag}*] \\
cpli_{frag} & ::= & cpli & | \ \underline{\text{lineFragment}}\ \text{isAntiFrag num [id*] } [e_{frag}*] \\
e_{frag} & ::= & e & | \ \underline{\text{entryFragment}}\ \text{isAntiFrag [id] [id]} \\
a_{frag} & ::= & a & | \ ia_{frag} \mid cia_{frag} \mid na_{frag} \\
ia_{frag} & ::= & ia & | \ \underline{\text{iarcFragment}}\ \text{id isAntiFrag id [id] [id]} \\
cia_{frag} & ::= & cia & | \ \underline{\text{ciarcFragment}}\ \text{id isAntiFrag id } [cit_{frag}] \\
na_{frag} & ::= & na & | \ \underline{\text{arcFragment}}\ \text{id isAntiFrag id [id] [id]} \\
cit_{frag} & ::= & cit & | \ \underline{\text{tableFragment}}\ \text{isAntiFrag } [cil_{frag}*] \\
cil_{frag} & ::= & cil & | \ \underline{\text{lineFragment}}\ \text{isAntiFrag num [id*] [id] [id]} \\
\text{isAntiFrag} & ::= & & \text{boolean}
\end{array}
$$

FIGURE 7.12 – Grammaire d'arbre des *HL*-LCP-frags.

7.3.3 Sémantique opérationnelle : règles de production

Nous exploitons maintenant les primitives d'une machine virtuelle LCP-net basique (telle que définie ultérieurement dans la section 9.2.2) afin d'établir la sémantique opérationnelle structurelle des *HL*-LCP-nets et de leurs fragments *via* un ensemble de règles de production qui établissent les correspondances entre leur représentation abstraite (cf. grammaires de la section 7.3.2) et les appels à ces primitives.

Concrètement, ces règles de production permettent de traduire tout *HL*-LCP-net en LCP-net, en vue de son évaluation dans la machine virtuelle. Concernant les *HL*-LCP-frags, les règles expriment la construction de leur fragments de plus bas niveau correspondant, ainsi que le contexte de leur application aux LCP-nets.

Avant de donner le détail de ces règles, il est nécessaire de définir la variable ρ qui correspond à l'environnement dans lequel ces règles sont évaluées. Plus précisément, $\rho : id \rightarrow V$ est une fonction *partielle* des identifiants sur les valeurs. Etant partielle, l'opération $\rho[id_x \rightarrow V_y]$ permet de définir la fonction pour l'identifiant id_x auquel on fait correspondre la valeur V_y.

Par ailleurs, dans les règles de production, on manipule les listes d'arguments ($arg*$) à l'aide des opérateurs usuels sur liste *cons*, *car* et *cdr*, représentés respectivement par les symboles $\oplus$, $\downarrow$ et $\dagger$. On étend aussi la définition de la fonction ρ de manière à pouvoir traiter les listes d'identifiants de cette manière : $\rho : (id \rightarrow v) \cup (id* \rightarrow V*)$.

Règles de production des *HL*-LCP-nets

$$\frac{\begin{array}{c}<\rho,\ \mathrm{ud}> \rightarrow <\rho_1,\ \mathrm{UD}> \\ <\rho_1,\ \mathrm{vd}*> \rightarrow <\rho_2,\ \mathrm{VD}*> \\ \mathrm{L} = \mathit{createLCPnet}(\text{“anonymous”},\ \mathrm{UD},\ \mathrm{VD}*) \\ <\rho_2,\ \mathrm{L},\ \mathrm{n}*> \rightarrow \rho_3 \\ <\rho_3,\ \mathrm{L},\ \mathrm{a}*> \rightarrow \rho_4\end{array}}{<\rho,\ \underline{\mathrm{lcpnet}}\ \mathrm{ud}\ \mathrm{vd}*\ \mathrm{n}*\ \mathrm{a}*> \rightarrow \rho[\mathrm{currentLCPnet} \rightarrow \mathrm{L}]}$$

Règle 7.1 – Règle de production d'un LCP-net.

$$\frac{<\rho,\ \mathrm{L},\ \downarrow \mathrm{n}*> \rightarrow \rho_1 \quad <\rho_1,\ \mathrm{L},\ \dagger \mathrm{n}*> \rightarrow \rho_2}{<\rho,\ \mathrm{L},\ \mathrm{n}*> \rightarrow \rho_2}$$

Règle 7.2 – Règle de parcours récursif de la liste des nœuds.

$$\frac{\mathrm{N} = \mathit{createLNode}(\mathrm{id}_1,\ \mathrm{L},\ \rho(\mathrm{id}_2)) \quad <\rho,\ \mathrm{N},\mathrm{cpt}> \rightarrow \rho_1}{<\rho,\ \mathrm{L},\ \underline{\mathrm{node}}(\mathrm{id}_1,\ \mathrm{id}_2,\ \mathrm{cpt})> \rightarrow \rho_1[\mathrm{id}_1 \rightarrow \mathrm{N}\ ]}$$

Règle 7.3 – Règle de production d'un nœud.

$$\frac{\mathrm{T} = \mathit{createCLPT}(\text{“table”},\ \mathrm{N}) \quad <\rho,\ \mathrm{T},\ \mathrm{cpli}*> \rightarrow \rho_1}{<\rho,\ \mathrm{N},\ \underline{\mathrm{cptable}}\ \mathrm{cpli}*> \rightarrow \rho_1}$$

Règle 7.4 – Règle de production d'une table de préférences.

$$\frac{<\rho,\ \mathrm{T},\ \downarrow \mathrm{cpli}^*> \rightarrow \rho_1 \quad <\rho_1,\ \mathrm{T},\ \dagger \mathrm{cpli}^*> \rightarrow \rho_2}{<\rho,\ \mathrm{T},\ \mathrm{cpli}^*> \rightarrow \rho_2}$$

Règle 7.5 – Règle de parcours récursif de la liste des lignes d'une table.

$$\frac{\mathrm{LI} = \mathit{createCLPTLine}(\text{“line”},\ \mathrm{T},\ \rho(\mathit{id}*)) \quad <\rho,\ \mathrm{LI},\ \mathrm{e}^*> \rightarrow \rho_1}{<\rho,\ \mathrm{T},\ \underline{\mathrm{line}}\ \mathrm{id}^*\ \mathrm{e}^*> \rightarrow \rho_1}$$

Règle 7.6 – Règle de production d'une ligne de table de préférence.

$$\frac{<\rho,\ \mathrm{LI},\ \downarrow \mathrm{e}^*> \rightarrow \rho_1 \quad <\rho_1,\ \mathrm{LI},\ \dagger \mathrm{e}^*> \rightarrow \rho_2}{<\rho,\ \mathrm{LI},\ \mathrm{e}^*> \rightarrow \rho_2}$$

Règle 7.7 – Règle de parcours récursif de la liste des entrées de la table.

$$\frac{\mathit{createEntry}(\text{“entry”},\ \mathrm{LI},\ \rho(\mathrm{id}_1),\ \rho(\mathrm{id}_2))}{<\rho,\ \mathrm{LI},\ \underline{\mathrm{entry}}\ \mathrm{id}_1\ \mathrm{id}_2> \rightarrow \rho}$$

Règle 7.8 – Règle de production d'une entrée de la table.

$$\frac{<\rho,\ \mathrm{L},\ \downarrow \mathrm{a}^*> \rightarrow \rho_1 \quad <\rho_1,\ \mathrm{L},\ \dagger \mathrm{a}^*> \rightarrow \rho_2}{<\rho,\ \mathrm{L},\ \mathrm{a}^*> \rightarrow \rho_2}$$

Règle 7.9 – Règle de parcours récursif de la liste des arcs.

$$\frac{\mathit{createArc}(\mathrm{id}_1,\ \mathrm{L},\ \rho(\mathrm{id}_2),\ \rho(\mathrm{id}_3))}{<\rho,\ \mathrm{L},\ \underline{\mathrm{arc}}\ \mathrm{id}_1\ \mathrm{id}_2\ \mathit{id}_3> \rightarrow \rho}$$

Règle 7.10 – Règle de production d'un arc.

$$\frac{\mathit{createIArc}(\mathrm{id}_1,\ \mathrm{L},\ \rho(\mathrm{id}_2),\ \rho(\mathrm{id}_3))}{<\rho,\ \mathrm{L},\ \underline{\mathrm{iarc}}\ \mathrm{id}_1\ \mathrm{id}_2\ \mathit{id}_3> \rightarrow \rho}$$

Règle 7.11 – Règle de production d'un i-arc.

$$\frac{\mathrm{CIA} = \mathit{createCIArc}(\mathit{id},\ \mathrm{L}) \quad <\rho,\ \mathrm{CIA},\ \mathrm{cit}>}{<\rho,\ \mathrm{L},\ \underline{\mathrm{ciarc}}\ \mathrm{id}\ \mathrm{cit}> \rightarrow \rho}$$

Règle 7.12 – Règle de production d'un ci-arc.

$$\frac{\text{CIT} = \textit{createCIT}(\text{“CIT”, CIA}) \quad <\rho, \text{CIT, cil*}>}{<\rho, \text{CIA, } \underline{\text{citable}} \text{ cil*}> \rightarrow \rho}$$

Règle 7.13 – Règle de production d'une table d'indépendance conditionnelle.

$$\frac{<\rho, \text{CIT}, \downarrow\text{cil*}> \rightarrow \rho_1 \quad <\rho_1, \text{CIT}, \dagger\text{cil*}> \rightarrow \rho_2}{<\rho, \text{CIT, cil*}> \rightarrow \rho_2}$$

Règle 7.14 – Règle de parcours récursif des lignes d'une CIT.

$$\frac{\textit{createCITLine}(\text{“line”, CIT, } \rho(id*), \text{ id}_1, \text{ id}_2)}{<\rho, \text{CIT, } \underline{\text{line}} \text{ id* id}_1 \text{ id}_2> \rightarrow \rho}$$

Règle 7.15 – Règle de production d'une ligne de CIT.

$$\frac{\text{D} = \textit{createLinguisticDomain}(\text{id, lb, ub}) \quad <\rho, \text{D, s*}> \rightarrow \rho_1}{<\rho, \underline{\text{utilityDomain}} \text{ id lb ub s*}> \rightarrow <\rho_1[\text{id} \rightarrow \text{D}], \text{D}>}$$

Règle 7.16 – Règle de production d'un domaine d'utilité.

$$\frac{<\rho, \text{D}, \downarrow\text{s*}> \rightarrow \rho_1 \quad <\rho_1, \text{D}, \dagger\text{s*}> \rightarrow \rho_2}{<\rho, \text{D, s*}> \rightarrow \rho_2}$$

Règle 7.17 – Règle de parcours récursif de la liste des sous-ensembles flous d'un domaine d'utilité.

$$\frac{\text{S} = \textit{createLinguisticSubset}(\text{id, D}) \quad <\rho, \text{S, c*}> \rightarrow \rho_1}{<\rho, \text{D, } \underline{\text{subset}} \text{ id c*}> \rightarrow \rho_1[\text{id} \rightarrow \text{S}]}$$

Règle 7.18 – Règle de production d'un SEF.

$$\frac{<\rho, \text{S}, \downarrow\text{c*}> \rightarrow \rho_1 \quad <\rho, \text{S}, \dagger\text{c*}> \rightarrow \rho_2}{<\rho, \text{S, c*}> \rightarrow \rho_2}$$

Règle 7.19 – Règle de parcours récursif de la liste des coordonnées d'un SEF.

$$\frac{\textit{createCoordinate}(\text{S}, \textit{real}_1, \textit{real}_2)}{<\rho, \text{S}, \underline{\text{coordinate}} \textit{ real}_1 \textit{ real}_2> \rightarrow \rho}$$

Règle 7.20 – Règle de production d'une coordonnée de SEF.

$$\frac{<\rho,\ \downarrow \mathrm{vd}^*> \rightarrow <\rho_1,\ \mathrm{VD}> \quad <\rho_1,\ \dagger \mathrm{vd}^*> \rightarrow <\rho_2,\ \mathrm{VD}_{reste}{}^*>}{<\rho,\ \mathrm{vd}^*> \rightarrow <\rho_2,\ \mathrm{VD} \oplus \mathrm{VD}_{reste}{}^*)>}$$

Règle 7.21 – Règle de production de la liste de tous les domaines de valeur.

$$\frac{\mathrm{D} = \mathit{createLinguisticDomain}(\mathrm{id},\ \mathrm{lb},\ \mathrm{ub}) \quad <\rho,\ \mathrm{D},\ \mathrm{s}^*> \rightarrow \rho_1}{<\rho,\ \underline{\mathrm{valueDomain}}\ \mathrm{id}\ \mathrm{lb}\ \mathrm{ub}\ \mathrm{s}^*> \rightarrow <\rho_1[\mathrm{id} \rightarrow \mathrm{D}],\ \mathrm{D}>}$$

Règle 7.22 – Règle de production d'un domaine de valeur spécifique.

Règles de production des *HL*-LCP-frags

Dans les règles de production ci-dessous, on effectue la distinction entre fragments et anti-fragments par le paramètre *isAntiFrag* défini sur la grammaire des *HL*-LCP-frags. En fonction de la valeur de ce booléen, la règle de production choisie n'est pas la même. On part aussi du principe que le LCP-net cible courant, auquel s'applique le fragment, est disponible dans son environnement ρ d'évaluation sous la dénomination *currentLCPnet*.

Par ailleurs, cette liste de règles n'est pas exhaustive car leur grande majorité peut être construite de manière automatique, à partir des règles de production des *HL*-LCP-nets. Le modèle à suivre consiste :

- à ajouter et traiter les paramètres spécifiques aux fragments : nom du fragment, booléen de distinction entre fragment et anti-fragment, référence à l'élément de base (ou élément cible) auquel le fragment doit s'appliquer,
- à remplacer les appels aux primitives de création d'éléments LCP-net de base, par des appels aux primitives spécifiques de création de fragments.

Ce modèle est illustré dans les règles de production suivantes :

$$\frac{\begin{array}{c}<\rho,\ \mathrm{ud}_{frag}> \rightarrow <\rho_1,\ \mathrm{UD}_{frag}> \\ <\rho_1,\ \mathrm{vd}_{frag}{}^*> \rightarrow <\rho_2,\ \mathrm{VD}_{frag}{}^*> \\ \mathrm{F} = \mathit{createLCPnetFragment}(\text{“anonymous”},\ \rho(\mathbf{currentLCPnet}),\ \mathrm{UD}_{frag},\ \mathrm{VD}_{frag}{}^*) \\ <\rho_2,\ \mathrm{F},\ \mathrm{n}_{frag}{}^*> \rightarrow \rho_3 \\ <\rho_3,\ \mathrm{F},\ \mathrm{a}_{frag}{}^*> \rightarrow \rho_4 \\ \mathrm{L} = \mathit{getCombinedLCPnet}(\mathrm{F})\end{array}}{<\rho,\ \underline{\mathrm{lcpnetFragment}}\ \mathrm{ud}_{frag}\ \mathrm{vd}_{frag}{}^*\ \mathrm{n}_{frag}{}^*\ \mathrm{a}_{frag}{}^*> \rightarrow \mathrm{L}}$$

Règle 7.23 – Règle de production et application directe d'un fragment de LCP-net.

$$\frac{D_{frag} = \mathit{createLinguisticDomainFragment}(\mathrm{id}_1,\ \mathit{false},\ \rho(\mathrm{currentLCPnet}).\mathit{getLinguisticDomain}(\mathrm{id}_2)\ ,\ \mathrm{lb},\ \mathrm{ub}) \quad <\rho,\ \mathrm{D}_{frag},\ \mathrm{s}_{frag}{}^*> \rightarrow \rho_1}{<\rho,\ \underline{\mathrm{utilityDomainFragment}}\ \mathrm{id}_1\ \mathit{false}\ \mathrm{id}_2\ \mathrm{lb}\ \mathrm{ub}\ \mathrm{s}_{frag}{}^*> \rightarrow <\rho_1[\mathrm{id}_1 \rightarrow \mathrm{D}_{frag}],\ \mathrm{D}_{frag}>}$$

Règle 7.24 – Règle de production d'un fragment de domaine d'utilité.

$$\frac{\mathrm{D}_{frag} = \mathit{createLinguisticDomainFragment}(\mathrm{id}_1,\ \mathit{true},\ \rho(\mathrm{currentLCPnet}).\mathit{getLinguisticDomain}(\mathrm{id}_2))}{<\rho,\ \underline{\mathrm{utilityDomainFragment}}\ \mathrm{id}_1\ \mathit{true}\ \mathrm{id}_2> \rightarrow <\rho[\mathrm{id}_1 \rightarrow \mathrm{D}_{frag}],\ \mathrm{D}_{frag}>}$$

Règle 7.25 – Règle de production d'un *anti-fragment* de domaine d'utilité.

$$\frac{<\rho,\ \mathrm{D}_{frag},\ \downarrow \mathrm{s}_{frag}{}^*> \rightarrow \rho_1 \quad <\rho_1,\ \mathrm{D}_{frag},\ \dagger \mathrm{s}_{frag}{}^*> \rightarrow \rho_2}{<\rho,\ \mathrm{D}_{frag},\ \mathrm{s}_{frag}{}^*> \rightarrow \rho_2}$$

Règle 7.26 – Règle de parcours récursif de la liste des fragments de sous-ensemble flous

$$\frac{\mathrm{S}_{frag} = \mathit{createLinguisticSubsetFragment}(\mathrm{id}_1,\ \mathit{false},\ \mathrm{D}_{frag}.\mathit{getBase}().\mathit{getSubset}(\mathrm{id}_2)\ ,\ \mathrm{D}_{frag}) \quad <\rho,\ \mathrm{S}_{frag},\ \mathrm{c}^*>}{<\rho,\ \mathrm{D}_{frag},\ \underline{\mathrm{subsetFragment}}\ \mathrm{id}_1\ \mathit{false}\ \mathrm{id}_2\ \mathrm{c}^*> \rightarrow \rho[\mathrm{id}_1 \rightarrow \mathrm{S}_{frag}]}$$

Règle 7.27 – Règle de production d'un fragment de sous-ensemble flou.

$$\frac{\mathrm{S}_{frag} = \mathit{createLinguisticSubsetFragment}(\mathrm{id}_1,\ \mathit{true},\ \mathrm{D}_{frag}.\mathit{getBase}().\mathit{getSubset}(\mathrm{id}_2)\ ,\ \mathrm{D}_{frag})}{<\rho,\ \mathrm{D}_{frag},\ \underline{\mathrm{subsetFragment}}\ \mathrm{id}_1\ \mathit{true}\ \mathrm{id}_2> \rightarrow \rho[\mathrm{id}_1 \rightarrow \mathrm{S}_{frag}]}$$

Règle 7.28 – Règle de production d'un *anti-fragment* de sous-ensemble flou.

$$\frac{<\rho,\ \mathrm{F},\ \downarrow n_{frag}*> \rightarrow \rho_1 \quad <\rho_1,\ \mathrm{F},\ \dagger n_{frag}*> \rightarrow \rho_2}{<\rho,\ \mathrm{F},\ n_{frag}*> \rightarrow \rho_2}$$

Règle 7.29 – Règle de parcours récursif de la liste des fragments de nœuds.

$$\frac{\mathrm{N}_{frag} = \mathit{createLNodeFragment}(\mathrm{id}_1,\ \mathit{false},\ \rho(\mathrm{currentLCPnet}).\mathit{getNode}(\mathrm{id}_2),\ \mathrm{F},\ \rho(\mathrm{id}_3)\ ?\rho(\mathrm{id}_3){:}\rho(\mathrm{currentLCPnet}).\mathit{getValueDomain}(\mathrm{id}_3)) \quad <\rho,\ \mathrm{N}_{frag},\ \mathrm{cpt}_{frag}> \rightarrow \rho_1}{<\rho,\ \mathrm{F},\ \underline{\mathrm{nodeFragment}}\ \mathrm{id}_1,\ \mathit{false},\ \mathrm{id}_2,\ \mathrm{id}_3,\ \mathrm{cpt}_{frag}> \rightarrow \rho_1[\mathrm{id}_1 \rightarrow \mathrm{N}_{frag}]}$$

Règle 7.30 – Règle de production d'un fragment de nœud.

$$\frac{\mathrm{N}_{frag} = \mathit{createLNodeFragment}(\mathrm{id}_1,\ \mathit{true},\ \rho(\mathrm{currentLCPnet}).\mathit{getNode}(\mathrm{id}_2),\ \mathrm{F})}{<\rho,\ \mathrm{F},\ \underline{\mathrm{nodeFragment}}\ \mathrm{id}_1,\ \mathit{true},\ \mathrm{id}_2> \rightarrow \rho_1[\mathrm{id}_1 \rightarrow \mathrm{N}_{frag}]}$$

Règle 7.31 – Règle de production d'un *anti-fragment* de nœud.

7.4 Intégration des *HL*-LCP-nets et *HL*-LCP-frags dans les processus métiers

Dans la section 7.2.2 nous avons étudié l'intérêt suscité par la possibilité de définir des réseaux de préférences utilisateur de manière incrémentale. Par la suite, nous avons défini la notion de fragment, introduit ses fondations au travers de la sémantique opérationnelle d'un langage LCP-net de haut niveau, et la syntaxe XML qui permet à un utilisateur d'exprimer de manière effective ses (fragments de) préférences.

Le but de cette section est alors d'étudier les possibilités d'intégration aux processus métiers des *HL*-LCP-nets et *HL*-LCP-frags, représentés en XML à un niveau de langage similaire à celui des processus BPEL. A cette fin nous examinerons les options suivantes : une première approche pour la définition et l'utilisation de préférences anonymes en portée lexicale, et une seconde approche où la manipulation des LCP-nets se rapproche de celle d'entités de plein droit (ou entités de "première classe"). Pour ces deux cas, nous proposerons une sémantique opérationnelle que nous limiterons volontairement aux activités d'intégration des *HL*-LCP-nets et *HL*-LCP-frags dans un processus BPEL. Une tentative de formalisation exhaustive de cette intégration nécessiterait la mise au point de règles de production pour l'ensemble des activités BPEL, ce qui sort du cadre de ce manuscrit.

7.4.1 Préférences anonymes en portée lexicale

Cette première option d'intégration permet, lors de l'évaluation d'une activité étendue *lateBindingInvoke* d'un processus BPEL, de disposer (s'il existe dans l'environnement d'exécution) d'un LCP-net *courant et anonyme*, auquel l'invocation de service est sujette et se réfère. Dans ce cas de figure, ce "facteur de préférence" est rattaché à une activité du processus métier dont la portée lexicale couvre, entre autres, cette activité d'invocation.

Les différentes activités de contrôle du flot d'exécution BPEL, bien que limitées, sont des candidates naturelles pour le rattachement d'un LCP-net "global" : *while*, *scope*, *sequence*, etc. On choisit, sur la figure 7.13, d'illustrer de manière graphique l'utilisation de la portée lexicale de l'activité *sequence* (en pointillés sur cette figure) : les activités n°1, 2 et 3 d'invocation de service sont toutes soumises à la préférence sur le temps de réponse T définie sur l'activité *BPEL:sequence*, par ailleurs, elles complètent cette préférence courante via leurs propres fragments de préférence $frag_1$, $frag_2$ et $frag_3$.

Sémantique opérationnelle

De manière à faire le pont entre les notions d'intégration de LCP-nets *via* la portée lexicale des activités BPEL, et la sémantique opérationnelle déjà établie de l'évaluation des *HL*-LCP-nets et *HL*-LCP-frags, on établit les règles de production suivantes :

– **Règle de production de la définition d'un *HL*-LCP-net** (cf. règle 7.32) : évaluée dans un environnement ρ similaire à celui des précédentes règles de production, complété par d'autres éléments de contexte liés à l'orchestration BPEL que l'on représentera par γ et que l'on ne formalisera pas. Il s'agit ici d'une définition rattachée à l'activité BPEL de séquence mais, comme nous l'avons évoqué, il existe d'autres possibilités. Elle dispose d'une liste d'activités

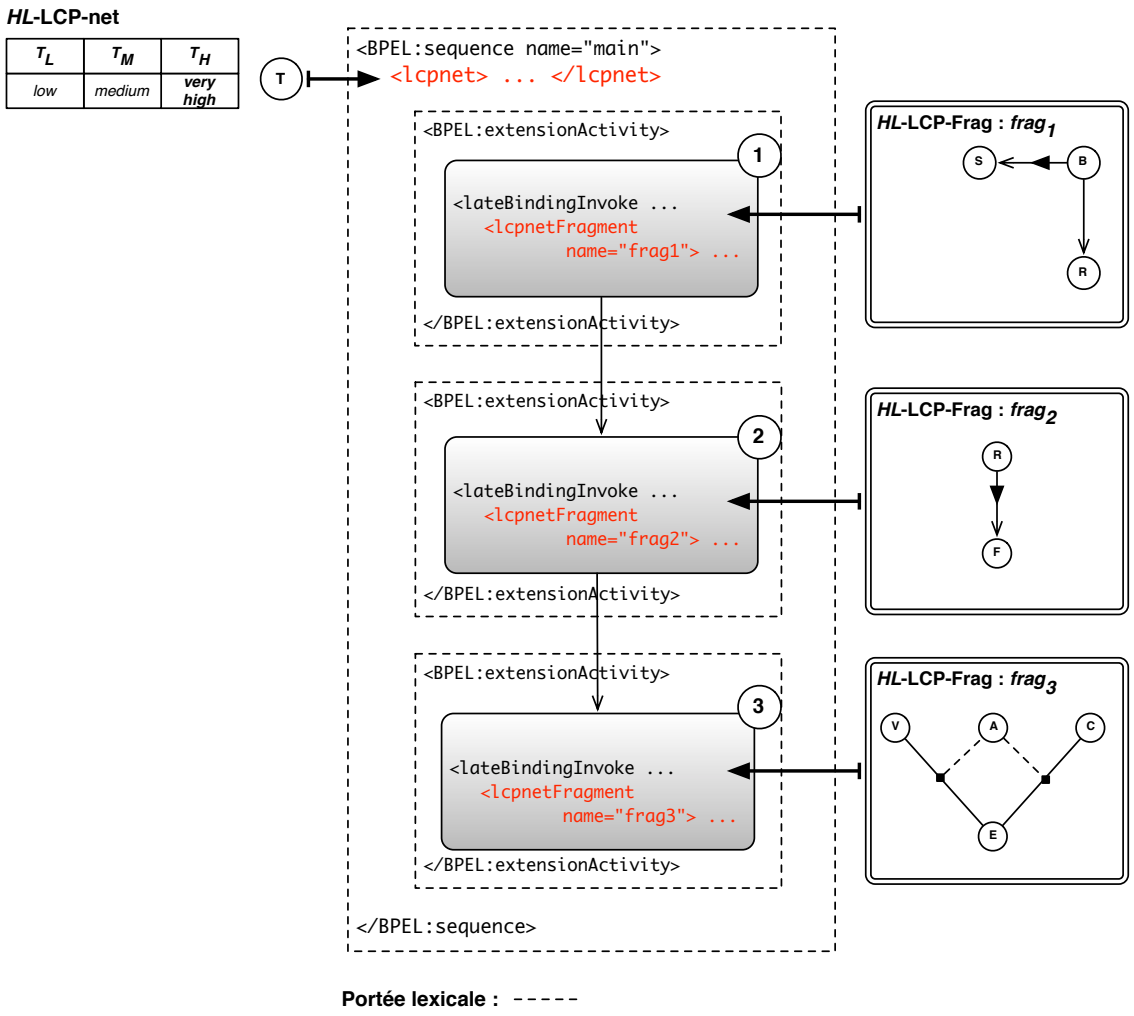

FIGURE 7.13 – Activité *lateBindingInvoke* dans le processus BPEL de gestion d'incendie.

(*activity* *), parmi lesquelles figurent les activités d'invocation en liaison tardive dont la règle de production est donnée ci-dessous. La règle 7.32 fait donc appel à la règle 7.1 de production d'un LCP-net. C'est dans l'environnement ρ_1 où le LCP-net a été produit (donc dans la portée lexicale de l'activité *BPEL:sequence*), que les activités de la séquence sont ensuite évaluées.

– **Règle de production de l'invocation en liaison tardive** (cf. règle 7.33) : l'environnement est défini avec ρ et γ selon le même principe que la règle 7.32. Cette règle signifie que l'invocation en liaison tardive de services avec un fragment de préférence *équivaut* à effectuer une invocation tardive où ce fragment a été préalablement appliqué à la préférence courante ($<\rho$, *fragment*$> \rightarrow L$, en lien avec la règle 7.23).

$$\frac{<\rho,\ \text{lcpnet}> \rightarrow \rho_1 \qquad <\rho_1,\ \gamma,\ \text{activity*}> \rightarrow \gamma_1}{<\rho,\ \gamma,\ \underline{\text{BPEL:sequence}}\ \text{lcpnet activity*}> \rightarrow \gamma_1}$$

Règle 7.32 – Règle de production de la définition d'un *HL*-LCP-net en portée lexicale.

$$\frac{<\rho,\ \text{fragment}> \rightarrow \text{L} \quad <\gamma,\ \underline{\text{lateBindingInvoke}}\ \text{L service* param*}> \rightarrow \gamma_1}{<\rho,\ \gamma,\ \underline{\text{lateBindingInvoke}}\ \text{fragment service* param*}> \rightarrow \gamma_1}$$

Règle 7.33 – Règle de production de l'invocation en liaison tardive avec portée lexicale.

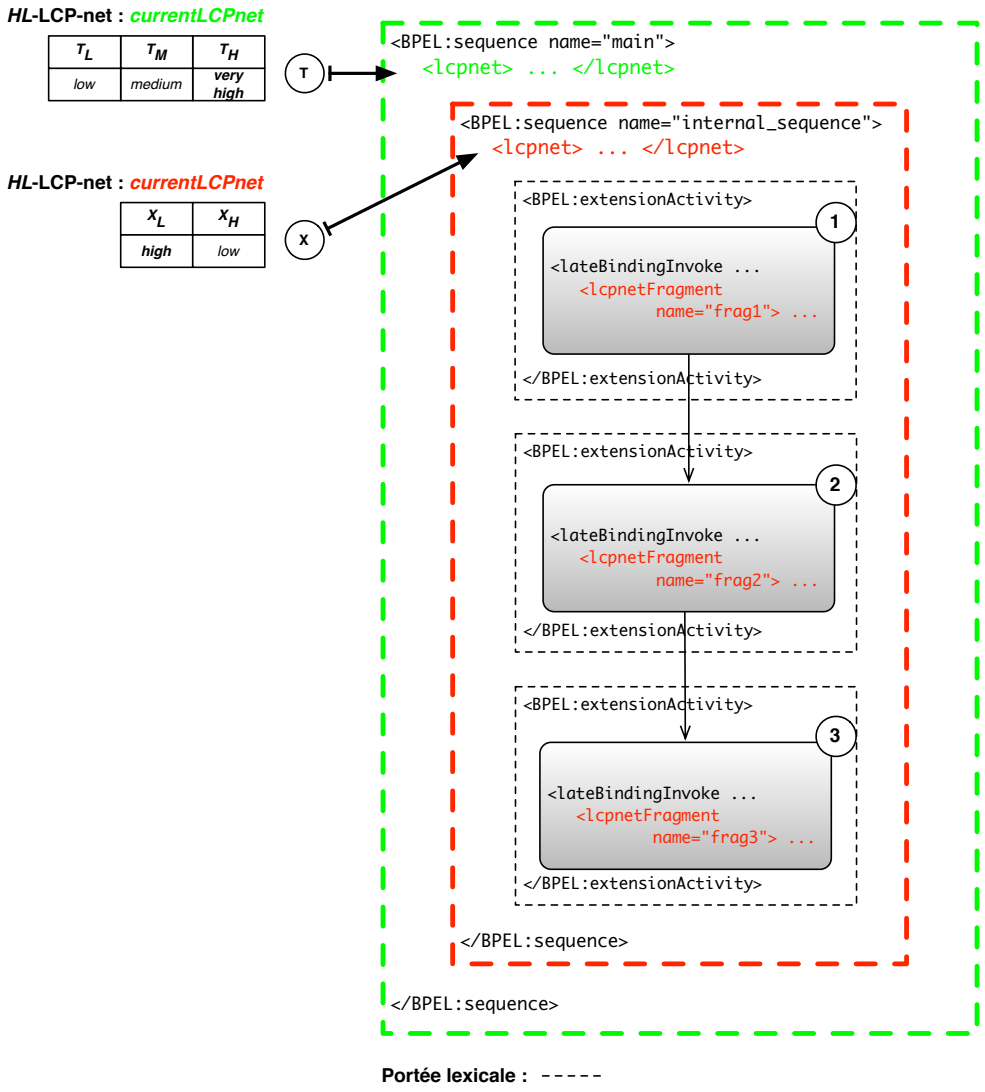

FIGURE 7.14 – Limitations de la portée lexicale.

7.4.2 Vers des LCP-nets entités de plein droit

Dans le cas précédent où la manipulation d'un LCP-net global repose sur une définition anonyme et sur la portée lexicale des activités BPEL de définition, il n'est possible de se référer, lors d'une invocation tardive de service, qu'à un unique LCP-net courant : en effet, seule la préférence rattachée à l'activité de contrôle du flot englobante n'a cours et masque les autres définitions. Ce principe est illustré dans la figure 7.14 : le LCP-net sur T rattaché à la séquence *main* est effectivement masqué par celui sur X rattaché à la séquence *internal_ sequence*, les activités n°1, 2 et 3 ne peuvent donc se référer qu'à ce dernier.

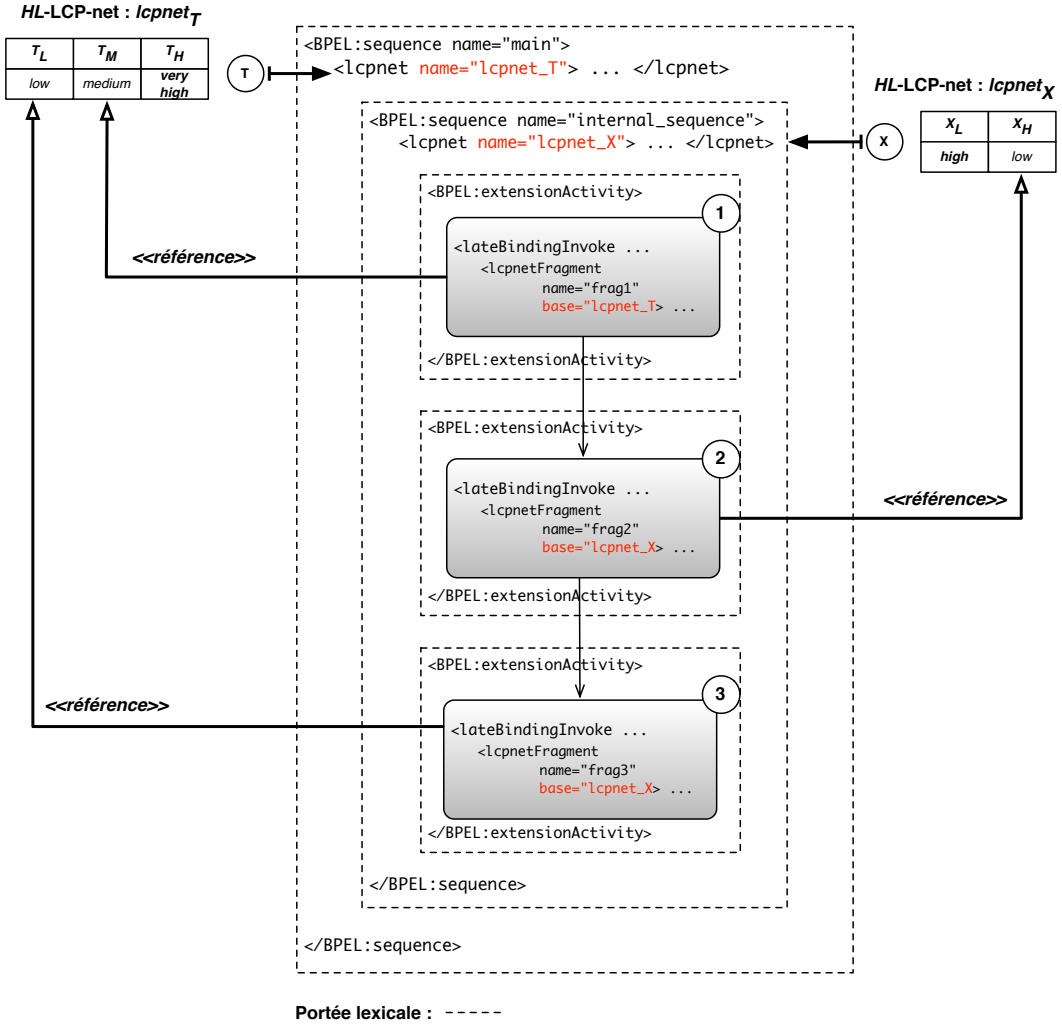

FIGURE 7.15 – LCP-nets comme entités de plein droit dans les processus BPEL.

Pour remédier à cette limitation, une première étape est de nommer explicitement les HL-LCP-nets et HL-LCP-frags lors de leur définition dans les processus BPEL. A ce titre, ils disposent d'un nom auquel il est possible de faire référence lors de l'invocation de service, tel qu'illustré par la figure 7.15.

Sémantique opérationnelle

Pour implanter ce nouveau paradigme d'utilisation, des modifications doivent cependant être appliquées aux règles de production des LCP-nets et de leurs fragments. En effet, elles reposent sur l'utilisation d'une entité *currentLCPnet*, disponible dans l'environnement ρ, de manière à pouvoir se référer au LCP-net courant anonyme à tout moment. Le nom propre des LCP-nets doit donc être maintenant considéré, et ces derniers stockés dans ρ sous leur désignation. On propose les nouvelles règles 7.34 et 7.35.

$$\frac{\begin{array}{c}<\rho,\ \mathrm{ud}> \rightarrow <\rho_1,\ \mathrm{UD}> \\ <\rho_1,\ \mathrm{vd}^*> \rightarrow <\rho_2,\ \mathrm{VD}^*> \\ \mathrm{L} = \mathit{createLCPnet}(\mathbf{id},\ \mathrm{UD},\ \mathrm{VD}^*) \\ <\rho_2,\ \mathrm{L},\ \mathrm{n}^*> \rightarrow \rho_3 \\ <\rho_3,\ \mathrm{L},\ \mathrm{a}^*> \rightarrow \rho_4\end{array}}{<\rho,\ \underline{\mathrm{lcpnet}}\ \mathbf{id}\ \mathrm{ud}\ \mathrm{vd}^*\ \mathrm{n}^*\ \mathrm{a}^*> \rightarrow \rho[\mathbf{id} \rightarrow \mathrm{L}]}$$

Règle 7.34 – Règle de production d'un LCP-net nommé.

$$\frac{\begin{array}{c}<\rho,\ \mathrm{ud}_{frag}> \rightarrow <\rho_1,\ \mathrm{UD}_{frag}> \\ <\rho_1,\ \mathrm{vd}_{frag}{}^*> \rightarrow <\rho_2,\ \mathrm{VD}_{frag}{}^*> \\ \mathrm{F} = \mathit{createLCPnetFragment}(\mathbf{id}_1,\ \rho(\mathbf{id}_2),\ \mathrm{UD}_{frag},\ \mathrm{VD}_{frag}{}^*) \\ <\rho_2,\ \mathrm{F},\ \mathrm{n}_{frag}{}^*> \rightarrow \rho_3 \\ <\rho_3,\ \mathrm{F},\ \mathrm{a}_{frag}{}^*> \rightarrow \rho_4 \\ \mathrm{L} = \mathit{getCombinedLCPnet}(\mathrm{F})\end{array}}{<\rho,\ \underline{\mathrm{lcpnetFragment}}\ \mathbf{id}_1\ \mathbf{id}_2\ \mathrm{ud}_{frag}\ \mathrm{vd}_{frag}{}^*\ \mathrm{n}_{frag}{}^*\ \mathrm{a}_{frag}{}^*> \rightarrow <\rho[\mathbf{id}_1 \rightarrow \mathrm{L}],\ \mathrm{L}>}$$

Règle 7.35 – Règle de production et application directe d'un fragment de LCP-net nommé.

On remarque alors que l'introduction d'un identifiant *name* pour désigner les fragments permet dorénavant de se référer au LCP-net résultant de son application sous le même identifiant *name* dans ρ. Cette extension ouvre donc la porte à des possibilités accrues de manipulation des entités dans les processus BPEL, telle qu'enchaîner les fragments "les uns derrières les autres" : par exemple un fragment ($frag_2$) de fragment ($frag_1$) de LCP-net ($lcpnet_{base}$). A ce titre, les préférences et leurs fragments se rapprochent d'entités de plein droit dans BPEL. Cependant, on peut s'interroger sur l'applicabilité d'une telle approche dans la pratique car il devient particulièrement difficile, au moment de l'écriture de $frag_2$, de savoir quels éléments du réseau $lcpnet_{base}$ sont encore désignables : ils auraient en effet très bien pu être supprimés préalablement par $frag_1$. Néanmoins, dans la mesure où les fragments se limitent à des ajouts d'éléments, cette problématique ne se pose alors plus, et les fragments conservent une très grande partie de leur intérêt.

Par ailleurs, une seconde étape vers la spécification des LCP-nets comme entités de plein droit au sein des processus serait de considérer les définitions de *HL*-LCP-nets comme des définitions de classes dont la visibilité ne serait plus limitée à la portée lexicale de l'activité BPEL de définition, mais à tout le processus.

7.5 QoS globale des processus et liaison tardive des services

Par rapport aux précédentes sections, les considérations que nous introduisons ici sont plus prospectives et procèdent d'une incarnation particulière au monde des SSOA de la notion de *"self monitoring"* [Gelenbe et al., 2004, Boloni et al., 1997]. Nous proposons en effet un moyen pour prendre en compte l'état courant des processus, au sein de leur propre exécution lors de la composition agile de services. Le but est d'être capable d'intégrer des informations sur la Qualité de Service globale courante des processus dans la décision de liaison tardive de services.

Le moyen que nous proposons pour réaliser cet objectif consiste à *exposer ces informations de QoS au travers des variables usuelles des LCP-nets*. En effet, le modèle de préférences est suffisamment souple pour permettre l'adjonction de ce type d'information sans modification préalable : du point de vue du modèle de préférences, une propriété sur la QoS globale du processus en cours d'exécution est de la même teneur que la QoS courante des services dont nous avons montré l'intégration précédemment (cf. chapitre 6). Les deux types d'information peuvent donc cohabiter dans le même modèle de préférences, et la QoS globale des processus influer la sélection de service, au même titre qu'une autre propriété, *en fonction des desiderata des utilisateurs*.

Des exemples typiques de propriétés globales de QoS auxquelles on souhaiterait pouvoir accéder sont le *temps écoulé* depuis le début de l'exécution du processus, la *bande passante disponible*, le *coût total* des services déjà liés, etc. On cherche donc le plus souvent à déterminer, de manière indirecte, les *ressources allouées au processus encore disponibles* et influer les futures sélections de service en conséquence. Un exemple de préférence utilisateur, au niveau d'un site d'appel de service, intégrant une propriété de ce type pourrait être :

"Si mon temps écoulé est encore faible, je préfère un service plus précis mais plus lent ; dans le cas contraire, je préfère un service moins précis mais qui impactera moins l'exécution du processus".

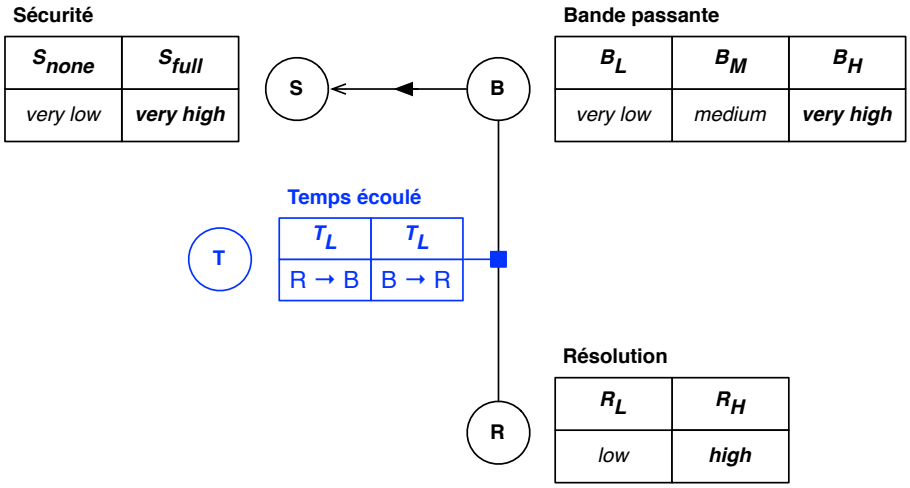

FIGURE 7.16 – LCP-net intégrant une dimension de QoS globale du processus métier.

Il est possible d'intégrer cette préférence sur la base de notre exemple de sélection d'un drone d'imagerie, où la résolution de l'image (donc la "*précision*") entre déjà en ligne de compte. Cette intégration est illustrée par la figure 7.16 : elle se traduit par une relation entre B et R de type *ci-arc*

qui inverse la préférence relative entre ces deux dimensions en fonction de la valeur d'une variable T qui représente le temps écoulé depuis le début de l'exécution du processus.

Ces dimensions impliquent toutes, à différents degrés, la mise au point d'un canevas de supervision dédié ou d'une extension d'un canevas existant. A l'heure actuelle, le bus de services Petals intégré dans le projet SemEUsE est en mesure d'effectuer une supervision au niveau de l'intergiciel SSOA et donc de fournir des mesures très basiques et bas niveau (le plus souvent techniques) sur l'état courant de l'orchestration. A terme, ces informations devraient être intégrées dans le canevas M4ABP [Le Duc et al., 2009] utilisé actuellement pour la supervision des services, de manière à, là aussi, rendre complètement transparente leur intégration dans le processus décisionnel de liaison actuel. Une piste sérieuse pour le calcul des propriétés courantes de QoS globale consisterait à le faire reposer sur les mêmes formules mathématiques que celles utilisées dans les travaux déjà existants sur la composition dynamique de services [Canfora et al., 2005b, Cardoso, 2002, Schröpfer et al., 2007], tels que vus dans l'état de l'art (cf. chapitre 3).

7.6 Conclusion

Dans ce chapitre, nous avons établi les abstractions de programmation nécessaires à l'utilisation conjointe de la composition *active* et *utile* de services. Cette approche donne naissance à la composition *agile*, qui présente ses propres subtilités.

On introduit ainsi la possibilité de définir ses préférences de manière incrémentale *via* la notion de *fragment* de préférence.

Il nous est aussi apparu utile de proposer une ingration plus poussée des préférences dans les processus métiers BPEL ; l'idée étant de tendre vers des LCP-nets considérés comme des entités de plein droit au cœur de ces processus. Cette constatation est à l'origine de l'introduction d'une représentation de plus haut niveau des LCP-nets et de leurs fragments : les *HL*-LCP-nets et *HL*-LCP-frags, dont une syntaxe XML permet une parfaite cohabitation avec les processus BPEL 2.0.

Grâce à la généricité de ce formalisme, il nous a été aussi possible d'intégrer des propriétés de *QoS globale* des processus dans les préférences, et donc, par la coordination agile, dans la décision de liaison elle-même. A plus long terme, des *données contextuelles* (telles que l'ampleur de l'incendie, sécheresse, etc.) ou des *QoS technique d'infrastructure* (fournies par le canevas SSOA) pourraient être intégrées tout aussi simplement, en suivant une méthodologie identique.

Chapitre 8

Formalisation du modèle LCP-net

Sommaire

8.1 Introduction

Dans cette thèse, nous avons proposé un modèle graphique pour représenter les préférences linguistiques (cf. chapitre 6) et l'avons déployé dans un cas d'utilisation spécifique (cf. chapitre 4). Nous consolidons ici cette contribution en la formalisant à travers un ensemble de notations et règles de calculs, afin d'en assurer la pérennité et la réutilisabilité à d'autres contextes de décision multi-critères.

8.2 Notations préliminaires

En s'inspirant des définitions des TCP-nets nous introduisons maintenant les LCP-nets de façon formelle. Soient :

- V_i une variable ($i \in \{1, \ldots, p\}$) : par exemple la sécurité,
- D_i le domaine de définition de V_i : par exemple $[0, 100]$,
- T_{V_i} l'ensemble de termes linguistiques associés à V_i : par exemple $\{S_{none}, S_{full}\}$,
- LV (une variable linguistique) définie comme étant le triplet suivant : $LV = \langle V, D, T_V \rangle$: par exemple $\langle$sécurité,$[0, 100]$,$\{S_{none}, S_{full}\}\rangle$.

Comme dans le formalisme UCP-nets, les préférences sont exprimées par des utilités, mais au travers de variables linguistiques, tout comme les autres variables. Elles prennent leurs valeurs dans le triplet unique $\langle V_U, D_U, T_{V_U} \rangle$ défini une fois pour toutes :

par exemple $\langle$utilité,$[0, 1]$,$\{$very_low,low,medium,high,very_high$\}\rangle$.

Une utilité est un triplet $LV_U = \langle V_U, D_U, S_{V_U} \rangle$, avec $S_{V_U} \in T_{V_U}$, par exemple $\langle$utilité,$[0,1]$,low$\rangle$.

Une table de préférences conditionnelles $CPT(LV)$ associe des préférences sur D pour toutes les affectations possibles aux parents de LV notés $Pa(LV)$. De plus, tout comme dans le formalisme TCP-nets, chaque arc non dirigé (arête) est annoté par une table d'importance conditionnelle $CIT(LV)$. Une CIT associée à une arête (LV_i, LV_j) décrit l'importance relative de LV_i par rapport à LV_j connaissant la valeur de la variable linguistique d'importance conditionnelle correspondante LV_k.

Graphiquement, une table de préférences (CPT ou CIT) est un tuple de triplets, i.e. une table à N dimensions. N est le nombre de variables linguistiques liées à LV, y compris LV ($N = |Pa(LV)|+1$) et une utilité S_{V_U} est définie dans chaque cas.

Ainsi, une table de préférences est représentée par le tuple

$\langle LV_{i_1}, LV_{i_2}, \ldots, LV_{i_N}, LV_{U_1}, LV_{U_2}, \ldots, LV_{U_\eta} \rangle$ avec $\eta = |T_{V_{i_1}}| \times |T_{V_{i_2}}| \times \ldots \times |T_{V_{i_N}}|$ et $\min(\eta) = 2 \times N$. $LV_{U_1}, LV_{U_2}, \ldots, LV_{U_\eta}$ sont les *mêmes* variables linguistiques d'utilité, mais dupliquées η fois.

Par exemple, une table de préférences peut être le tuple suivant :

$\langle\langle$résolution,$[320, 1024]$,$\{R_L, R_H\}\rangle$, $\langle$bande passante,$[56, 4096]$,$\{B_L, B_M, B_H\}\rangle$, $\langle$utilité,$[0,1]$,very_high$\rangle$, $\langle$utilité,$[0,1]$,very_low$\rangle$, $\langle$utilité,$[0,1]$,high$\rangle$, $\langle$utilité,$[0,1]$,low$\rangle$, $\langle$utilité,$[0,1]$,very_low$\rangle$, $\langle$utilité,$[0,1]$,very_high$\rangle\rangle$.

Plus précisément, une table de préférences est égale à

$$
\begin{array}{l}
\langle\ \langle S^1_{V_{i_1}}, S^1_{V_{i_2}}, \ldots, S^1_{V_{i_N}}, S^1_{V_U} \rangle \\
\quad \langle S^1_{V_{i_1}}, S^1_{V_{i_2}}, \ldots, S^2_{V_{i_N}}, S^2_{V_U} \rangle, \\
\quad \ldots \\
\quad \langle S^{|T_{V_{i_1}}|}_{V_{i_1}}, S^{|T_{V_{i_2}}|}_{V_{i_2}}, \ldots, S^{|T_{V_{i_N}}|}_{V_{i_N}}, S^{\eta}_{V_U} \rangle \\
\rangle
\end{array}
$$

avec $S^1_{V_i}$ le premier terme linguistique dans le partitionnement de la variable V_i.

Ainsi nous obtenons η tuples $\langle S_{V_{i_1}}, S_{V_{i_2}}, \ldots, S_{V_{i_N}}, S_{V_U} \rangle$. La raison pour laquelle le minimum pour η est égal à $2 \times N$ est parce qu'il est nécessaire que $|T_V| \geq 2$ pour être capable d'exprimer une préférence (!).

En suivant le même exemple et en considérant que la résolution et la bande passante sont liées entre elles, la table de préférences associée peut être définie par ces six ($\eta = 6$) tuples :

$\langle\langle R_L, B_L, very\_high \rangle$, $\langle R_L, B_M, very\_low \rangle$, $\langle R_L, B_H, high \rangle$, $\langle R_H, B_L, low \rangle$, $\langle R_H, B_M, very\_low \rangle$, $\langle R_H, B_H, very\_high \rangle\rangle$.

Ces tuples sont à voir comme six règles impliquant trois variables linguistiques différentes (deux LV plus la LV d'utilité), c'est-à-dire six fois un ensemble constitué de deux triplets $\langle V, D, S_V \rangle$ et d'un triplet $\langle V_U, D_U, S_{V_U} \rangle$:

R1. Si nous avons $\langle$résolution,$[320, 1024]$,$R_L\rangle$ et $\langle$bande passante,$[56, 4096]$,$B_L\rangle$ alors nous avons $\langle$utilité,$[0, 1]$,$very\_high\rangle$;

R2. ...

...

R6. Si nous avons $\langle$résolution,$[320, 1024]$,$R_H\rangle$ et $\langle$bande passante,$[56, 4096]$,$B_H\rangle$ alors nous avons $\langle$utilité,$[0, 1]$,$very\_high\rangle$.

8.3 Formalisme et test de dominance sur un LCP-net

Sur la base des notations précédentes, on définit alors un LCP-net $\mathcal{L}$ comme étant un tuple $\langle SL$,cp,i,ci,cpt,cit$\rangle$ où :

- SL est un ensemble de variables linguistiques $\{LV_1, \ldots, LV_p\}$,
  par exemple $SL = \{\langle$sécurité,$[0, 100]$,$\{S_{none}, S_{full}\}\rangle$, $\langle$bande passante,$[56, 4096]$,$\{B_L, B_M, B_H\}\rangle$, $\langle$résolution,$[320, 1024]$,$\{R_L, R_H\}\rangle\}$,
- cp est un ensemble de cp-arcs orientés. Un cp-arc $\langle\overrightarrow{LV_i, LV_j}\rangle$ est dans $\mathcal{L}$ si et seulement si les préférences sur les valeurs de LV_j dépendent de la valeur actuelle de LV_i. Pour chaque $LV \in SL$, $Pa(LV) = \{LV' | \langle\overrightarrow{LV', LV}\rangle \in$ cp$\}$,
- i est un ensemble d'i-arcs orientés. Un i-arc $(\overrightarrow{LV_i, LV_j})$ est dans $\mathcal{L}$ si et seulement si $LV_i \triangleright LV_j$,
- ci est un ensemble de ci-arcs non-orientés. Un ci-arc (LV_i, LV_j) est dans $\mathcal{L}$ si et seulement si nous avons $\mathcal{RI}(LV_i, LV_j | LV_k)$, i.e. si et seulement si l'importance relative de LV_i par rapport à LV_j est conditionnée par LV_k, avec $LV_k \in SL \setminus \{LV_i, LV_j\}$. Nous appelons LV_k le *sélecteur* de (LV_i, LV_j) et le notons $\mathcal{S}(LV_i, LV_j)$,
- cpt associe une CPT à chaque variable linguistique $LV \in SL$, où $CPT(LV)$ est une fonction de $D_{Pa(LV) \cup LV}$ (i.e., affectations aux variables linguistiques parentes de LV plus la LV considérée) dans D_U,
- cit associe à chaque ci-arc entre LV_i et LV_j une CIT qui est une fonction de $D_{\mathcal{S}(LV_i, LV_j)}$ dans l'ensemble des relations d'ordre sur $\{LV_i, LV_j\}$.

Les CPT (attachées à une LV) fournissent une utilité locale pour cette LV. Cette utilité locale notée lu est calculée grâce à un moteur d'inférence utilisant les règles évoquées plus haut.

Ainsi, d'un côté, nous obtenons un LCP-net qui définit les préférences et de l'autre, p utilités locales notées de la façon suivante :

$$LU = \bigcup_{i=1}^{p} \{lu_i\}$$

Chaque nœud (que l'on peut associer à une LV) de $\mathcal{L}$ est ensuite affecté d'un poids w, on obtient donc un vecteur de poids W :

$$W = \bigcup_{i=1}^{p} \{w_i\}$$

$\mathcal{L}$ peut ainsi désormais être représenté par le tuple $\langle SL,\mathsf{cp},\mathsf{i},\mathsf{ci},\mathsf{cpt},\mathsf{cit},W\rangle$, sachant que les valeurs prises par W dépendent de la profondeur des nœuds.

L'algorithme de calcul de W peut être fondé sur une fonction de la famille BUM (Basic Unit-interval Monotonic). Une fonction BUM f_{BUM} est définie de $[0,1]$ dans $[0,1]$ et admet les propriétés suivantes :

- $f_{BUM}(0) = 0$
- $f_{BUM}(1) = 1$
- f_{BUM} est croissante (càd si $x > y$ alors $f_{BUM}(x) \geq f_{BUM}(y)$)

Les poids W sont donc calculés grâce à f_{BUM} de la façon suivante :

$$w_i = f_{BUM}(i/p) - f_{BUM}((i-1)/p)$$

La fonction f_{BUM} choisie peut être $f_{BUM}(x) = x$ (dans ce cas, tous les poids sont égaux à $(1/p)$ avec p le nombre de nœuds) ; ou $f_{BUM}(x) = x^3$ (dans ce cas, w_1 est très petit par rapport à w_p) ; ou encore $f_{BUM}(x) = \sqrt{x}$ (dans ce cas, w_1 est le plus grand poids). Pour analyser le choix de f_{BUM}, on peut calculer la mesure d'*orness* sur ce vecteur de poids :

$$orness(W) = \frac{1}{p-1} \sum_{i=1}^{p} (p-i) w_i$$

Cette mesure, comprise entre 0 et 1, permet d'exprimer à quel point l'agrégateur utilisant ces poids ressemble à un OU. Par exemple, quand $f_{BUM}(x) = x$, $orness(W) = 0.5$. Mais lorsque w_1 est très grand par rapport aux poids "suivants", $orness(W)$ tend vers 1.

Comme dans les CP-nets, plus on descend dans la profondeur des nœuds, et moins ils sont importants : on choisira donc un vecteur W dont la mesure *orness* est comprise entre 0.5 et 1 [15], c'est-à-dire $f_{BUM}(x) = \sqrt{x}$ ou $\sqrt[3]{x}$, etc.

Affecter des poids aux nœuds d'un graphe est quelque peu différent d'une affectation classique de poids à des valeurs. En effet, la différence réside dans l'ordonnancement de ces valeurs. Dans un graphe LCP-net, plusieurs nœuds peuvent avoir la même profondeur, donc l'ordre n'est pas total. C'est pourquoi, affecter seulement des w à l'aide d'une fonction BUM, même judicieusement choisie, ne permet pas de répondre complètement à notre problème, puisque les nœuds de même profondeur seront discriminés (des poids différents leur seront affectés). Donc nous appliquons une fonction BUM telle que les w associés obtenus soient décroissants ($w_i > w_{i+1}$, avec $i \in [1,p]$). Puis, pour chaque nœud de même profondeur, on somme les poids qui leur sont associés et on effectue une équirépartition de cette somme entre ces nœuds. Ainsi, toutes les contraintes sont respectées, par construction des poids *via* f_{BUM} :

15. Dans notre implémentation, le vecteur de poids obtenu (cf. section 6.4) vérifie bien ce critère.

– $\sum_{i=1}^{p} w_{i,l_i} = 1$ avec l_i le niveau de profondeur du nœud i, $l_i \in [1, L]$ et $L \leq p$

– $\forall i \in [1,p], \forall l_i \in [1,L], \begin{cases} w_{i,l_i} > w_{i+1,l_{i+1}} \text{ si } l_i \neq l_{i+1} \\ w_{i,l_i} = w_{i+1,l_{i+1}} \text{ sinon} \end{cases}$

W est combiné avec LU pour obtenir l'utilité globale associée à une affectation o noté GU_o.

$GU_o = \Delta(LU_o, W)$, avec Δ un agrégateur pondéré, type OWA [Yager, 1988] (par exemple une simple moyenne pondérée).

Une utilité locale est soit un terme linguistique, soit un nombre correspondant à la défuzzification (par l'opérateur d) du SEF : $lu = f_{S_{V_U}}$ ou $lu = d(f_{S_{V_U}})$ avec :

$$f_{S_{V_U}} = f_{S_{V_U}}(y) = \begin{cases} \bot(f_{S^1_{V_U}}(y), \ldots, f_{S^\eta_{V_U}}(y)) & \text{si les } \eta \text{ règles sont indépendantes} \\ \top(f_{S^1_{V_U}}(y), \ldots, f_{S^\eta_{V_U}}(y)) & \text{sinon} \end{cases}$$

avec $y \in D_U$, $\bot$ une t-conorme et $\top$ une t-norme.

Pour simplifier, posons $lu_i = f_{S_{V_{U_i}}}(y)$. En conséquence, GU_o est soit un terme linguistique, soit un nombre. Dans le cas où c'est un terme linguistique, il est toujours possible de trouver un opérateur de défuzzification d qui fournit (donc) un nombre.

Considérant que les règles sont indépendantes entre elles, on obtient, en appliquant le modus ponens généralisé, l'égalité suivante :

$$f_{S_{V_U}}(y) = \sup_{(x_1,\ldots,x_N)\in D_1\times\cdots\times D_N} \Big\{ \top \Big[g(f_{S_{V'_1}}(x_1), \ldots, f_{S_{V'_N}}(x_N)), \Phi(g(f_{S^1_{V_1}}(x_1), \ldots, f_{S^1_{V_N}}(x_N)), f_{S^1_{V_U}}(y)) \Big] \\ \vee \ldots \vee \top \Big[g(f_{S_{V'_1}}(x_1), \ldots, f_{S_{V'_N}}(x_N)), \Phi(g(f_{S^\eta_{V_1}}(x_1), \ldots, f_{S^\eta_{V_N}}(x_N)), f_{S^\eta_{V_U}}(y)) \Big] \Big\}$$

avec $f(x)$ la fonction d'appartenance de l'élément x, Φ une implication floue, V' les variables réelles observées (notons bien que la notation "prime" n'a pas le même sens que plus haut : ici, on a N variables *observées* notées de V'_1 à V'_N), $S_{V'_1}$ le terme linguistique associé à la première variable observée (V'_1) et g un opérateur d'agrégation comme une t-norme (min par exemple). Ainsi, une affectation o est en fait un tuple $\langle S_{V'_1}, \ldots, S_{V'_p} \rangle$.

Afin de prouver la propriété de *test de dominance*, nous montrons qu'une affectation o peut toujours être trouvée comme étant strictement préférée à une autre affectation o'.

Théorème.
Etant donné un LCP-net $\mathcal{L}$ et une paire d'affectations o et o', $\mathcal{L} \models o \prec o'$ si et seulement si GU_o est plus petit que $GU_{o'}$.

Preuve.

$$\frac{\text{inférence calculée par } CPT(LV_1^o) \quad \ldots \quad \text{inférence calculée par } CPT(LV_p^o)}{\mathcal{L} \vdash lu_1^o = f_{S_{V_{U,1}^o}} \quad \ldots \quad lu_p^o = f_{S_{V_{U,p}^o}}}$$

$$\frac{\mathcal{L} \vdash lu_1^o = f_{S_{V_{U,1}^o}} \quad \ldots \quad lu_p^o = f_{S_{V_{U,p}^o}}}{\mathcal{L} \vdash LU_o = \{lu_1^o, \ldots, lu_p^o\}}$$

$$\frac{\text{inférence calculée par } CPT(LV_1^{o'}) \quad \ldots \quad \text{inférence calculée par } CPT(LV_p^{o'})}{\mathcal{L} \vdash lu_1^{o'} = f_{S_{V_{U,1}^{o'}}} \quad \ldots \quad lu_p^{o'} = f_{S_{V_{U,p}^{o'}}}}$$

$$\frac{\mathcal{L} \vdash lu_1^{o'} = f_{S_{V_{U,1}^{o'}}} \quad \ldots \quad lu_p^{o'} = f_{S_{V_{U,p}^{o'}}}}{\mathcal{L} \vdash LU_{o'} = \{lu_1^{o'}, \ldots, lu_p^{o'}\}}$$

$$\frac{\mathcal{L} \vdash LU_o = \{lu_1^o, \ldots, lu_p^o\} \qquad \mathcal{L} \vdash LU_{o'} = \{lu_1^{o'}, \ldots, lu_p^{o'}\}}{\mathcal{L} \vdash \Delta(LU_o, W) < \Delta(LU_{o'}, W)}$$

$$\frac{\mathcal{L} \vdash \Delta(LU_o, W) < \Delta(LU_{o'}, W)}{\mathcal{L} \vdash d(GU_o) < d(GU_{o'})}$$

$$\frac{\mathcal{L} \vdash d(GU_o) < d(GU_{o'})}{\mathcal{L} \vdash GU_o \prec GU_{o'}}$$

$$\frac{\mathcal{L} \vdash GU_o \prec GU_{o'}}{\mathcal{L} \models o \prec o'}$$

□

8.4 Conclusion

S'il est possible de représenter graphiquement un LCP-net comme un ensemble de nœuds, d'arcs et de CPTs, ce chapitre nous a montré que, mathématiquement, c'est un tuple constitué de LVs, de trois types d'arcs, de deux types de tables et d'un vecteur de poids. De surcroît, un LCP-net se présente après traduction comme un ensemble de LVs, de FIS et de poids associés aux nœuds qui autorisent un raisonnement automatique sur ces préférences et posent les bases de leur mise en œuvre.

Chapitre 9

Mise en œuvre

Sommaire

9.1 Introduction

Ce chapitre présente la mise en œuvre de nos contributions en termes de *composition active* et *utile* de services [Châtel et al., 2010a], dont le déploiement conjoint constitue les fondations techniques d'une véritable *composition agile* de services. Elle est effectuée au travers de l'implantation Java d'un canevas de modélisation et décision sur LCP-nets (cf. section 9.2), ainsi que de l'intégration de la notion de liaison tardive au sein d'un moteur d'orchestration spécifique de services Web (cf. section 9.3).

Plusieurs choix ont dû être effectués pour mener à bien notre travail de mise en œuvre ; ils seront détaillés au cours du chapitre, ainsi que l'ensemble des informations techniques nécessaires à leur bonne compréhension.

9.2 Canevas de modélisation et décision sur LCP-nets

Dans cette section nous effectuons un panorama des considérations techniques liées à la mise en œuvre d'un canevas de modélisation et décision sur LCP-nets, mais aussi et surtout des choix qu'il a été nécessaire d'effectuer pour le réaliser, en fonction des contraintes liées au contexte.

De manière à traiter d'emblée les plus générales de ces considérations, il apparaît utile de préciser que l'implantation du canevas a été réalisée dans sa totalité en Java. Le choix de ce langage s'est effectué naturellement de par le contexte SSOA des travaux de cette thèse où il prédomine, mais aussi pour des raisons de facilité d'intégration dans le cadre plus général du projet SemEUsE. Ce choix est également justifié par l'utilisation, au sein même du canevas, de composants Java : en particulier le moteur d'inférence floue retenu, qui offre une API entièrement Java.

Par ailleurs, les développements ont été effectués sur la plate-forme Eclipse : nous avons mis à profit son éditeur graphique ("IDE"), mais surtout certaines des bibliothèques gravitant autour du projet Eclipse, telles qu'EMF, qui a grandement facilité la mise au point de notre propre éditeur graphique pour les préférences LCP-nets.

Enfin, l'ensemble du code et des modèles relatifs à l'implantation du canevas est accessible publiquement sur `http://code.google.com/p/lcp-nets` sous licence GPLv3 : l'éditeur graphique ainsi que le moteur d'inférence sur LCP-nets y sont tout deux disponibles au téléchargement.

Dans les sous-sections suivantes nous abordons successivement le modèle de données des préférences (cf. 9.2.1), l'outillage des LCP-nets (cf. 9.2.2), qui permet notamment leur création incrémentale *via* les primitives d'une machine virtuelle, les principes régissant l'implantation du calcul et de la décision sur LCP-nets (cf. 9.2.3) ainsi que l'application des fragments (cf. 9.2.2), avant de procéder, pour conclure, à un bref rappel des différents choix d'implantation (cf. 9.2.4).

9.2.1 Modèle de données structuré des préférences et des fragments

L'ensemble des éléments constitutifs d'un LCP-net ou d'un fragment, tels qu'ils ont été présentés jusqu'à présent dans ce manuscrit, ont été modélisés grâce au *canevas EMF* [Budinsky et al., 2003]. Constitué d'une bibliothèque et d'un outil de modélisation et génération de code, il est intimement lié à Eclipse, sa principale particularité étant de proposer un méta-modèle dont le niveau d'abstraction est à mi-chemin entre un diagramme de classe UML classique et le code Java. Il présente en ce sens une alternative de modélisation simple mais puissante, à forte visée implantatoire, tout en permettant aussi l'import et la conversion de modèles UML classiques, lorsque justifié par l'ampleur du projet. Dans le cadre de notre implantation, *nous avons choisi d'utiliser directement le méta-modèle EMF* dont l'expressivité offerte s'est révélée suffisante pour nos besoins.

Cette modélisation EMF a été établie de manière à englober non-seulement la notion de LCP-net que nous introduisons dans ce manuscrit, mais aussi celle de CP-net "générique" telle que définie par Boutilier *et al.* [Boutilier et al., 2004], d'UCP-net [Boutilier et al., 2001] et de TCP-net [Brafman et Domshlak, 2002]. Ce choix a été effectué, plutôt que de se focaliser uniquement sur les LCP-nets, de manière à *faciliter les extensions au canevas que nous mettons au point.* Il nous a par ailleurs permis de réaliser, en sus de celle des LCP-nets, une implantation fonctionnelle de la modélisation et de la décision sur UCP-nets. Cette modélisation globale est par ailleurs une bonne illustration de la proche parenté de ces formalismes.

Modèle *ecore* des données

Un modèle de données des LCP-nets a donc été établi sous la forme d'un fichier *ecore*, propre à EMF, et sérialisé selon le standard XMI de l'OMG. De par son ampleur, nous choisissons d'en présenter ici une vue partielle, tout d'abord sous la même forme arborescente que celle accessible à

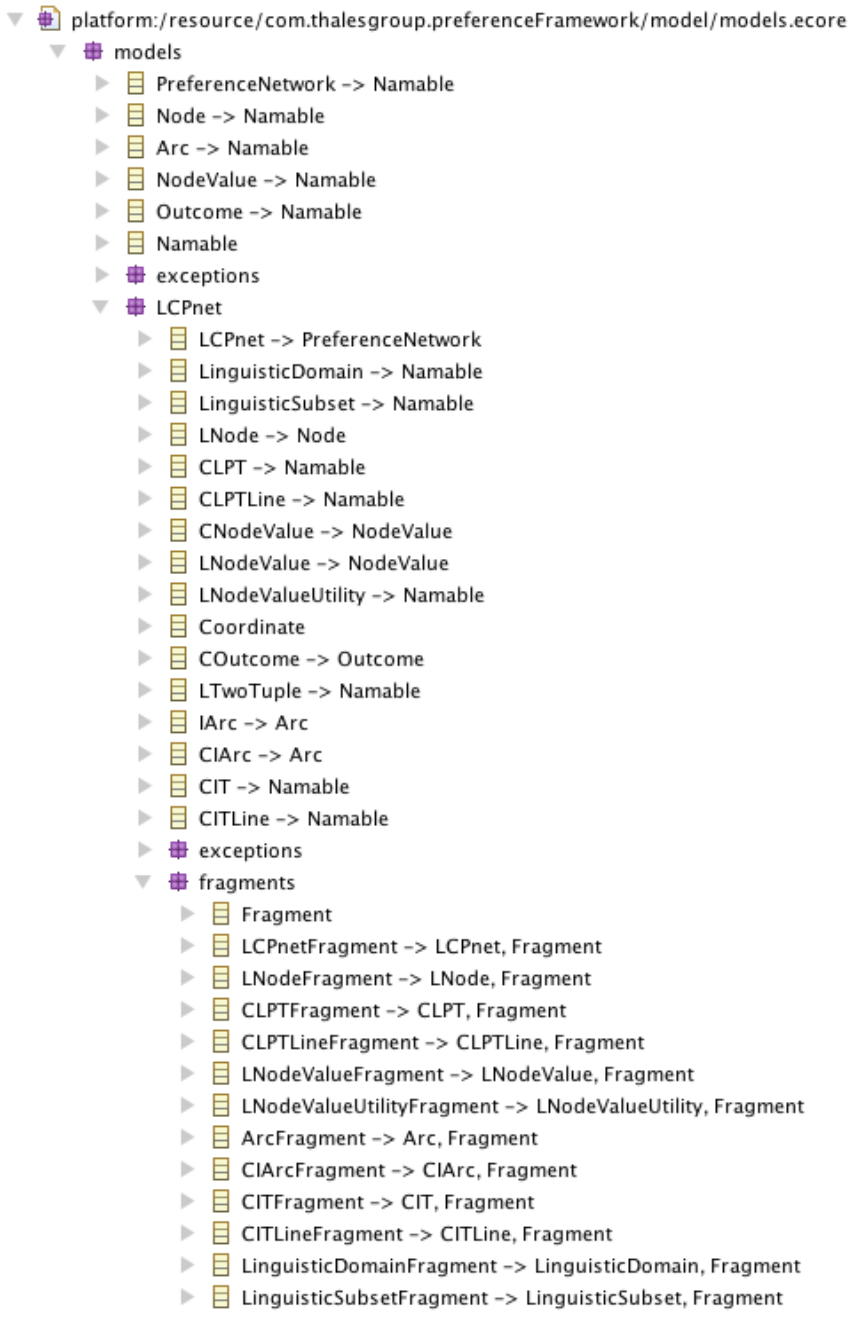

FIGURE 9.1 – Représentation arborescente partielle du modèle de données LCP-net.

travers l'IDE Eclipse à la figure 9.1. On remarque que les éléments constitutifs du modèle, ou *EClass* selon la terminologie Eclipse, ont été répartis, en fonction de leur appartenance, sous trois paquets ("packages") distincts et imbriqués, de manière à atteindre une saine séparation des concepts : sous *'models'* l'ensemble des concepts généraux aux quatre formalismes (notion de réseau de préférences, de nœud, etc.), sous *'models/LCPnet'* ceux plus spécifiques aux LCP-nets (Domaine linguistique,

table linguistique de préférences, etc.) et sous *'models/LCPnet/fragments'* les éléments constitutifs des fragments de LCP-nets.

Lors de la modélisation, nous avons utilisé la *relation d'héritage entre éléments EClass*, similaire à la relation d'héritage Java, pour pratiquer une factorisation du modèle et favoriser sa souplesse d'utilisation. Cette relation est illustrée dans la précédente figure par une flèche "->" indiquée sur les *EClass* : par exemple, *LCPnet* hérite de *PreferenceNetwork*, ainsi que de toutes ses propriétés qui ne figurent pas sur ce diagramme.

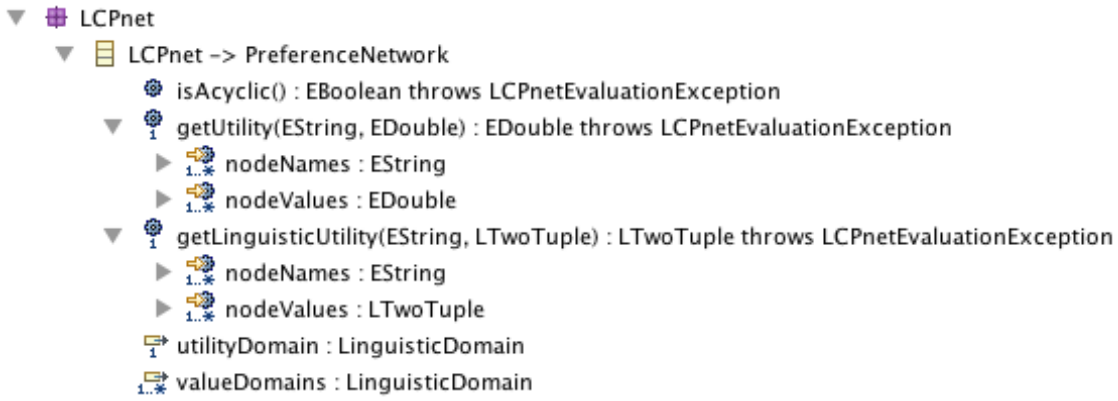

FIGURE 9.2 – Représentation arborescente de l'élément *LCPnet*.

On étend justement l'élément *LCPnet* de la précédente représentation arborescente sur la figure 9.2, de manière à exposer :

- **ses propriétés :** le partitionnement flou d'un domaine pour les valeurs d'utilité *utilityDomain* utilisé dans les tables, l'ensemble *valueDomains* des domaines de valeurs des nœuds du réseau de préférences ; auxquelles s'ajoutent les propriétés héritées de *PreferenceNetwork* et qui ne sont donc pas visibles : en particulier la liste des nœuds et celle des arcs du réseau ;

- **ses méthodes "métiers" et leurs arguments :** *isAcyclic()*, *getUtility(EString, EDouble)* et *getLinguisticUtility(EString, LTwoTuple)*. Nous choisissons en effet, selon une certaine vision de la conception objet, de *rattacher directement aux classes du modèle les méthodes de calcul et de décision sur leurs instances* (des méthodes d'instance métiers). Le détail de ces méthodes sera abordé dans la section 9.2.2. Par ailleurs, leur présence au sein même du modèle de données des LCP-nets témoigne bien du niveau d'abstraction intermédiaire d'un modèle EMF, par rapport à une vision UML et au code.

Il est à noter que des propriétés et méthodes idoines sont aussi définies sur les autres éléments du modèle EMF.

Cette vue arborescente est complétée par les figures 9.3 et 9.4, qui décrivent respectivement le graphe des relations entre concepts généraux et entre concepts spécifiques aux LCP-nets. Il est possible d'y voir qu'un réseau de préférences *PreferenceNetwork* est notamment composé d'arcs et de nœuds et que tous ses éléments héritent de l'*Eclass Namable*, ce qui signifie qu'il définissent tous une propriété permettant d'identifier leurs instances. Ces deux figures n'illustrent cependant pas la relation d'héritage qui existe *entre* les deux domaines conceptuels.

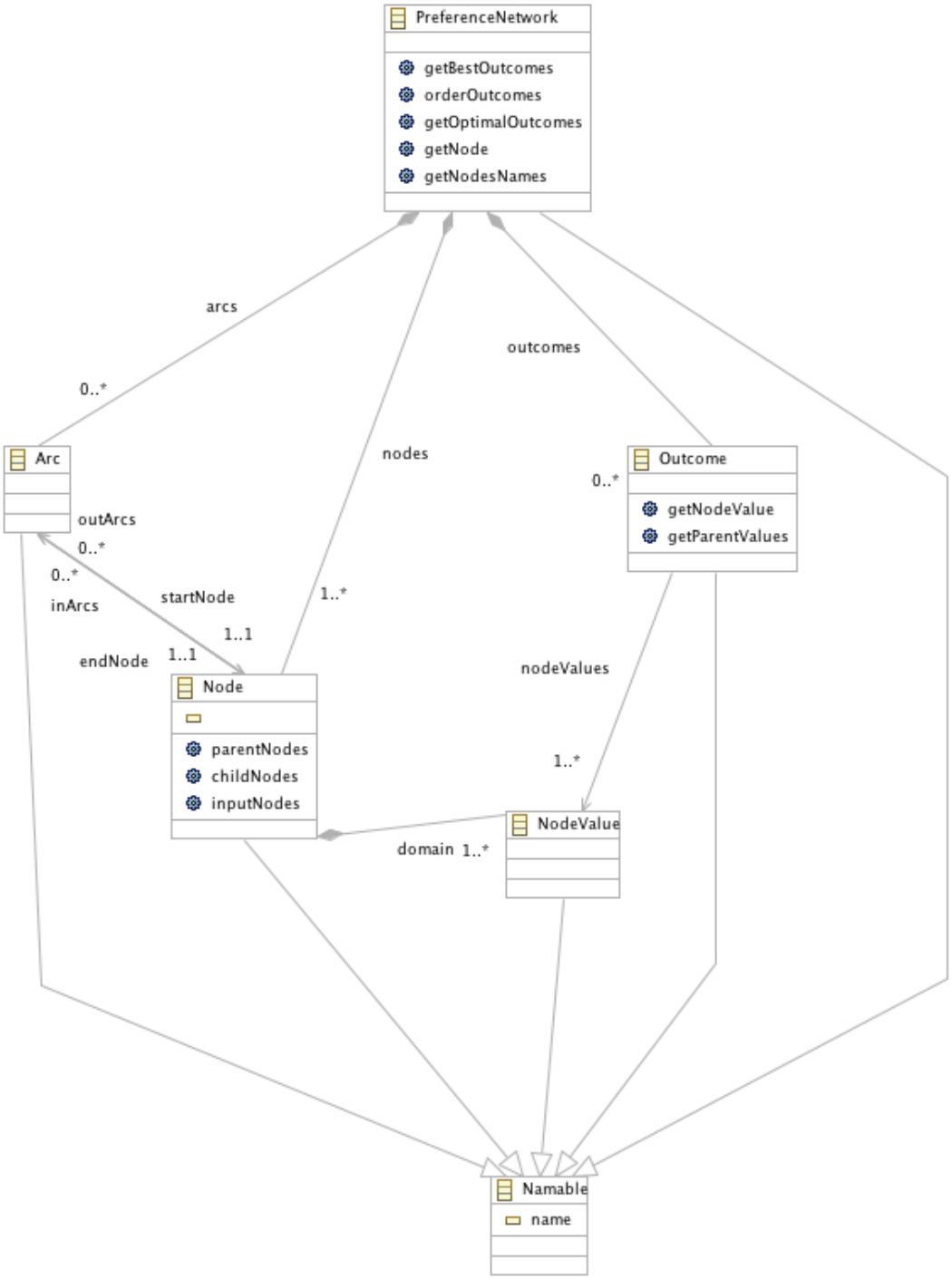

FIGURE 9.3 – Représentation graphique partielle du modèle de données LCP-net : *concepts généraux*

D'un modèle *ecore* vers une représentation Java

A partir de cette description *ecore* et des outils EMF à disposition, un *modèle objet* basique a pu être aisément obtenu :

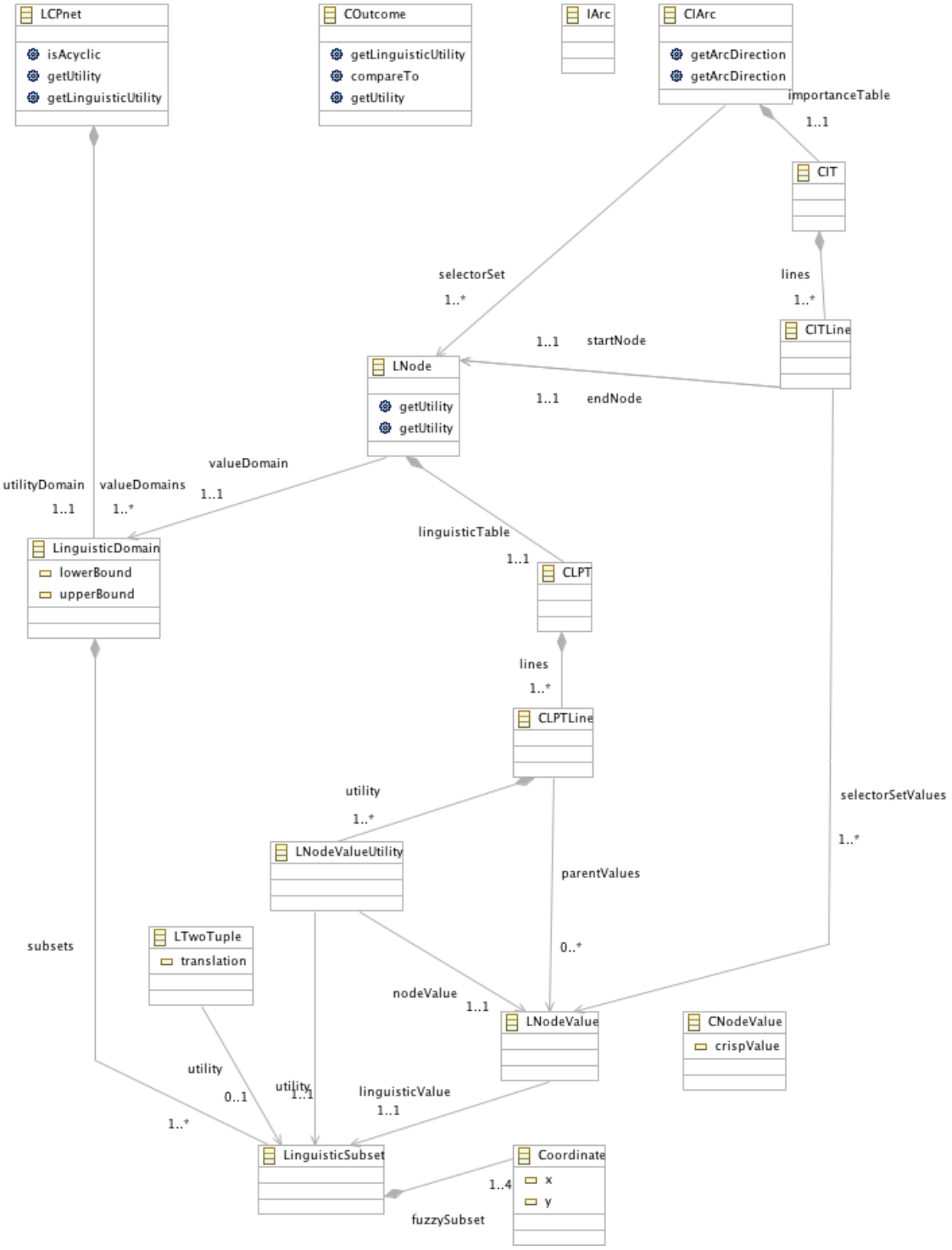

FIGURE 9.4 – Représentation graphique partielle du modèle de données LCP-net : *concepts spécifiques*

- à chaque *Eclass* du modèle est associé une interface Java du même nom, cette interface définit l'ensemble des accesseurs et modificateurs des propriétés spécifiées dans le modèle, en suivant le patron de conception d'encapsulation des données ;
- à chaque interface Int_x ainsi générée est associée, dans un sous-paquet *impl*, une classe Int_xImpl qui définit les propriétés propres à cet élément ainsi que leur typage extrait des informations contenues dans le modèle ;
- chacune de ces classes est en fait un squelette, dont seule la partie relative aux propriétés est pré-complétée. Il reste bien évidemment à fournir l'ensemble des informations qui n'ont pu être inférées du modèle de données EMF : en particulier tout ce qui concerne les méthodes "métier" des éléments du modèle ;
- pour chaque instance créée à partir de ces classes, le canevas EMF assure de manière automatique, après une configuration appropriée, leur sérialisation sous forme de fichiers XML. Des exemples de cette sérialisation ont été précédemment donnés par les fragments de code 6.1 et 7.4. Elles font notamment appel à des expressions XPath pour gérer la désignation des entités.

Il faut noter que le code généré suit le *patron de conception "factory"* dans la mesure où il délègue les opérations de création d'instances à des classes dédiées qui les regroupent par domaine : à titre d'exemple, toutes les opérations d'instanciation des éléments propres aux LCP-net sont ainsi regroupées dans une classe *(...).models.LCPnet.LCPnetFactory* par lequel l'utilisateur doit passer, et non *via* les constructeurs propres aux classes.

Par ailleurs, dans cette représentation sous forme de classes Java, *la relation d'héritage est naturellement transposée*. Ainsi, en correspondance directe avec le modèle, à chaque type LCP-net de base correspond un type de fragment (et donc une classe) qui en hérite. Par exemple, un fragment de nœud *LNodeFragment* est avant tout un *LNode* et dispose des mêmes propriétés, en sus de celles qui lui sont propres. *Ce qui permet d'affirmer sans ambiguïté qu'un fragment de LCP-net est effectivement un LCP-net lui-même et que sa construction va suivre fidèlement celle de son archétype*. Par ailleurs, les opérations de création d'un fragment de LCP-net ne dévieront que très peu de celle des LCP-nets puisque, en suivant le principe d'héritage, un fragment peut être utilisé partout où un type LCP-net de base était attendu.

9.2.2 Outillage des LCP-nets

Le canevas que nous avons mis au point assure deux fonctions essentielles : la modélisation de préférences LCP-net ainsi que la prise de décisions fondées sur ces modèles. Nous proposons un outillage spécifique pour l'accompagner.

Tout d'abord, pour permettre la création de modèles LCP-nets mais aussi de leur fragments, nous proposons un *regroupement logique* d'opérations élémentaires portant sur le modèle objet de données précédemment décrit. Ces opérations ne se limitent pas à la création de modèles, elles sont complétées par une méthode *getCombinedLCPnet()* permettant d'appliquer un fragment à un LCP-net, et surtout par une opération *getUtility()* d'inférence de la valeur d'utilité globale d'une affectation de valeurs à tous les nœuds d'un réseau de préférences. On considère alors que ce regroupement logique constitue une machine virtuelle (*"Virtual Machine"* ou *VM*) sur LCP-nets, *il s'agit du premier outil à la disposition des utilisateurs de ce canevas*. Il est présenté plus en détails ci-dessous.

Ensuite, la modélisation de préférence est accessible, non seulement au travers des primitives de cette machine virtuelle, mais aussi de manière graphique grâce à un outil d'édition de LCP-nets que nous avons mis au point. Les principes régissant l'utilisation de cet outil, qui peut être distribué sous forme de plugin Eclipse, sont décrits ci-après.

Primitives d'une machine virtuelle LCP-net

Le squelette d'une API exhaustive de création et manipulation de LCP-nets a donc été obtenu, à partir de la modélisation EMF des préférences, et complété pour le rendre fonctionnel. De par le niveau d'abstraction initial relativement bas du modèle EMF, cette API comporte un certain nombre de détails techniques d'implantation qui sont de peu d'intérêt pour l'utilisateur final.

De manière à l'isoler de ces couches basses de l'implantation, simplifier l'utilisation de l'API, et la présenter *comme un "tout" cohérent*, nous en détaillons ici un sous-ensemble qui inclut l'ensemble des primitives considérées comme nécessaires à la mise en œuvre d'une *machine virtuelle LCP-net*.

Cette notion de machine virtuelle prend tout son sens lorsqu'elle est mise en relation avec le langage d'expression de préférences haut niveau *HL*-LCP-nets et *HL*-LCP-frags, dont la syntaxe XML a été introduit au chapitre 7. Une métaphore possible serait en effet de comparer *HL*-LCP-nets et *HL*-LCP-frags à des fichiers sources *.java*, à mettre en opposition avec des fichiers compilés *.class*, qui seraient ici équivalents aux sérialisations XML produites par notre canevas. Par ailleurs, certaines des autres fonctionnalités que l'on est en mesure d'attendre d'une "véritable" machine virtuelle, telles que la gestion automatique de la mémoire, sont ici sous-traitées par la machine virtuelle Java elle-même (en l'occurrence par la mise en place du ramasse-miettes ou "Garbage Collector").

Ces opérations de base, ou primitives, permettent la construction progressive de LCP-nets, leur interrogation, la définition de fragments de LCP-nets (en suivant un modèle similaire à celle des LCP-nets) et la composition d'un fragment avec un LCP-net. La figure 9.5 illustre le cas d'utilisation où les primitives de construction de la VM sont d'abord mises à contribution pour obtenir un LCP-net "compilé" à partir d'un *HL*-LCP-net et *HL*-LCP-frag, ensuite pour réaliser le calcul d'utilité d'un LCP-net en fonction des valeurs de ces nœuds. Il faut noter que l'"utilisateur" qui y apparaît est défini au sens le plus large, c'est-à-dire qu'il pourrait tout aussi bien s'agir d'un programme informatique (par exemple l'éditeur graphique de préférences présenté ci-après) qu'un véritable utilisateur final humain. La VM est donc impliquée dans l'ensemble de ces activités de base.

La majeure partie de ces primitives, dont la liste est donnée ci-dessous, est tout droit extraite de l'API de base, les autres correspondent à une simplification des paramètres des méthodes initiales et sont implantées par l'adjonction d'un simple code "de colle" entre elles et l'API.

- **Primitives de partitionnement flou des domaines de valeurs :**
  1. `LinguisticDomain` *`createLinguisticDomain`*`(String name, Double lowerBound, Double upperBound) ;`
  2. `LinguisticSubset` *`createLinguisticSubset`*`(String name, LinguisticDomain domain) ;`
  3. `Coordinate` *`createCoordinate`*`(LinguisticSubset linguisticSubset, Double x, Double y) ;`
- **Primitives de construction des éléments LCP-nets de base :**

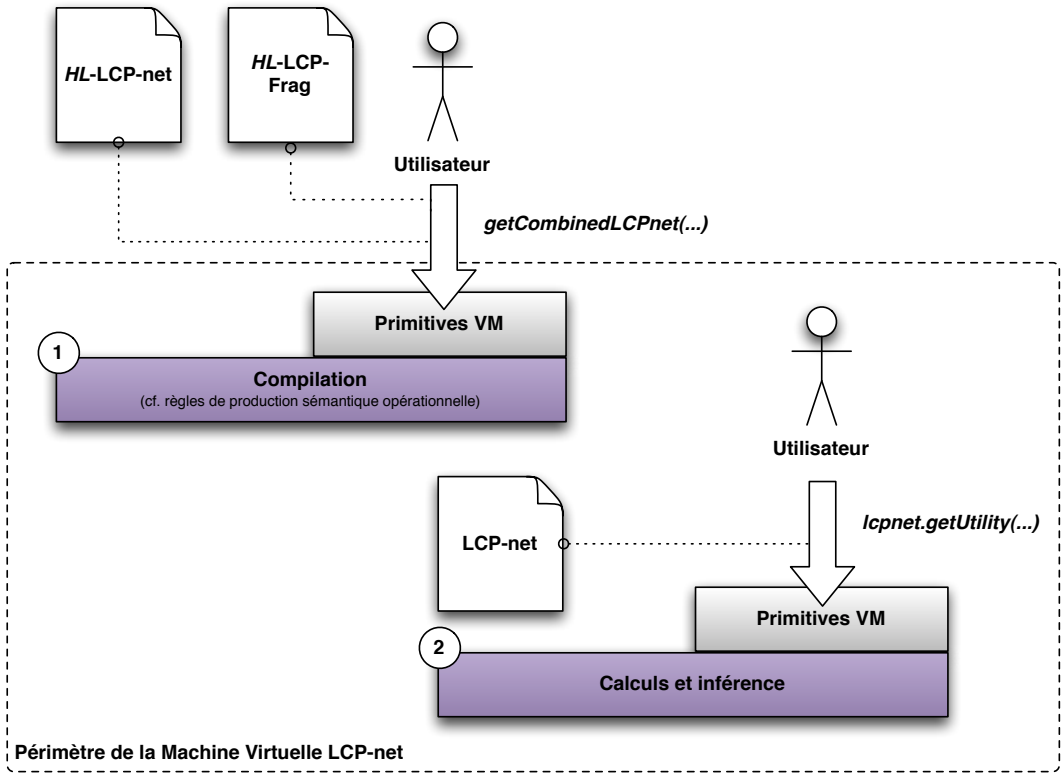

FIGURE 9.5 – Cas d'utilisation de la machine virtuelle LCP-net

1. `LCPnet` *`createLCPnet`*`(String name, LinguisticDomain utilityDomain, List<LinguisticDomain> valueDomains) ;`
2. `LNode` *`createLNode`*`(String name, LCPnet preferenceNetwork, LinguisticDomain valueDomain) ;`
3. `CNodeValue` *`createCNodeValue`*`(String name, LNode node, Double crispValue) ;`
4. `Arc` *`createArc`*`(String name, LCPnet preferenceNetwork, LNode startNode, LNode endNode) ;`
5. `IArc` *`createIArc`*`(String name, LCPnet preferenceNetwork, LNode startNode, LNode endNode) ;`
6. `CIArc` *`createCIArc`*`(String name, LCPnet preferenceNetwork) ;`

– **Primitives de construction des tables de préférences conditionnelles (*CLPT* et *CIT*) :**

1. `CLPT` *`createCLPT`*`(String name, LNode node) ;`
2. `CLPTLine` *`createCLPTLine`*`(String name, CLPT table, List<LinguisticSubset> parentValues) ;`
3. `void` *`createEntry`*`(String name, CLPTLine line, LinguisticSubset nodeValue, LinguisticSubset utility) ;`
4. `CIT` *`createCIT`*`(String name, CIArc arc) ;`

5. `CITLine` *`createCITLine`* `(String name, CIT table, List<LinguisticSubset> selectorSetValues, LNode startNode, LNode, endNode) ;`

– **Primitives de construction des fragments de préférence :**

1. `LCPnetFragment` *`createLCPnetFragment`* `(String fragmentName, LCPnet baseLCPnet, LinguisticDomain utilityDomain, List<LinguisticDomain> valueDomains) ;`
2. `LNodeFragment` *`createLNodeFragment`* `(String fragmentName, Boolean antiFragment, LNode baseLNode, LCPnet preferenceNetwork, LinguisticDomain valueDomain) ;`
3. `CLPTFragment` *`createCLPTFragment`* `(String fragmentName, Boolean antiFragment, CLPT baseCLPT, LNode node) ;`
4. `CLPTLineFragment` *`createCLPTLineFragment`* `(String fragmentName, Boolean antiFragment, CLPTLine base CLPTLine, CLPT table, List<LinguisticSubset> parentValues) ;`
5. `void` *`createEntryFragment`* `(String fragmentName, Boolean antiFragment, CLPTLine line, LinguisticSubset nodeValue, LinguisticSubset utility) ;`
6. `ArcFragment` *`createArcFragment`* `(String fragmentName, Boolean antiFragment, Arc baseArc, LCPnet preferenceNetwork, LNode startNode, LNode endNode) ;`
7. `CITFragment` *`createCITFragment`* `(String fragmentName, Boolean antiFragment, CIT baseCIT, CIArc arc) ;`
8. `CITLineFragment` *`createCITLineFragment`* `(String fragmentName, Boolean antiFragment, CITLine baseCITLine, CIT table, List<LinguisticSubset> selectorSetValues, LNode startNode, LNode, endNode) ;`
9. `LinguisticDomainFragment` *`createLinguisticDomainFragment`* `(String fragmentName, Boolean antiFragment, LinguisticDomain baseLinguisticDomain, Double lowerBound, Double upperBound) ;`
10. `LinguisticSubsetFragment` *`createLinguisticSubsetFragment`* `(String fragmentName, Boolean antiFragment, LinguisticSubset baseLinguisticSubset, LinguisticDomain domain) ;`

– **Primitive de composition d'un LCP-net avec un fragment :**
`LCPnet` *`getCombinedLCPnet`* `(LCPnetFragment) throws LCPnetCombinationException ;`
Nb : le LCP-net de base n'est pas indiqué dans les paramètres de la méthode car il est référencé directement par le fragment.

– **Primitive d'évaluation d'un LCP-net à partir des valeurs de ses nœuds :**
`Double` *`getUtility`* `(LCPnet preferenceNetwork, List nodeNames, List nodeValues) throws LCPnetEvaluationException ;`

Il est important de noter qu'après création d'un modèle objet de préférence (type *LCPnet* ou *LCPnetFragment*), obtenu en valeur de sortie des primitives des créations relatives, il est possible d'accéder, *a posteriori*, aux sous-éléments du modèle qui lui ont été ajoutés. Cet accès s'effectue indirectement par le biais *d'accesseurs* sur cet objet, de manière à respecter les propriétés d'encapsulation des données. Ces accesseurs reprennent le nom des propriétés concernées : par exemple, *LCPnet.getNode(String name)* pour obtenir un objet de type *node* déjà construit, à partir de son

nom (tous les éléments du modèle de préférences héritant de la classe *Namable*, ils disposent tous d'un identifiant).

Editeur graphique de préférences et fragments

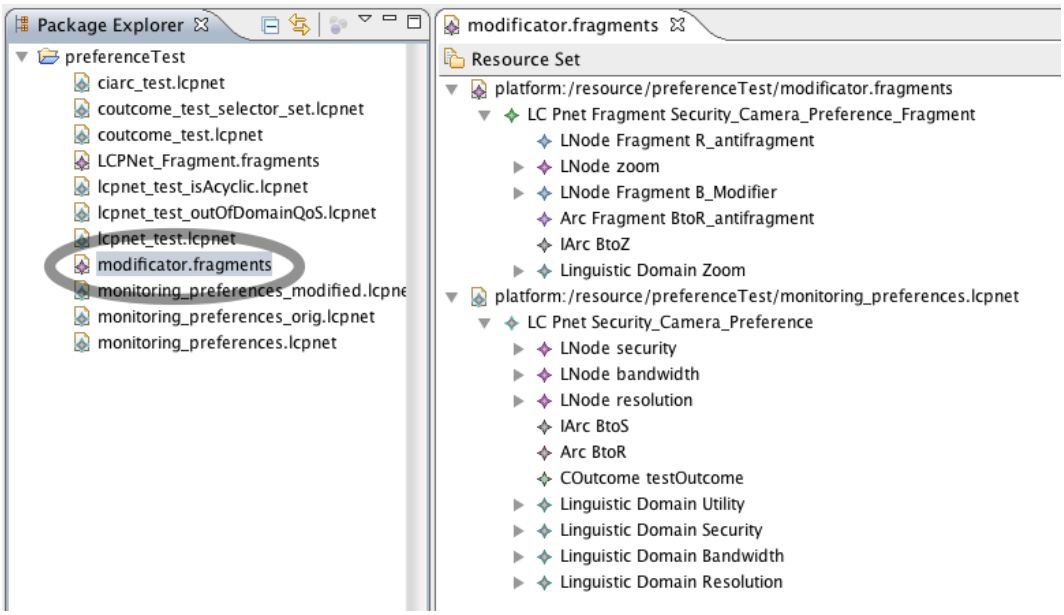

FIGURE 9.6 – Editeur de LCP-nets : vue complète.

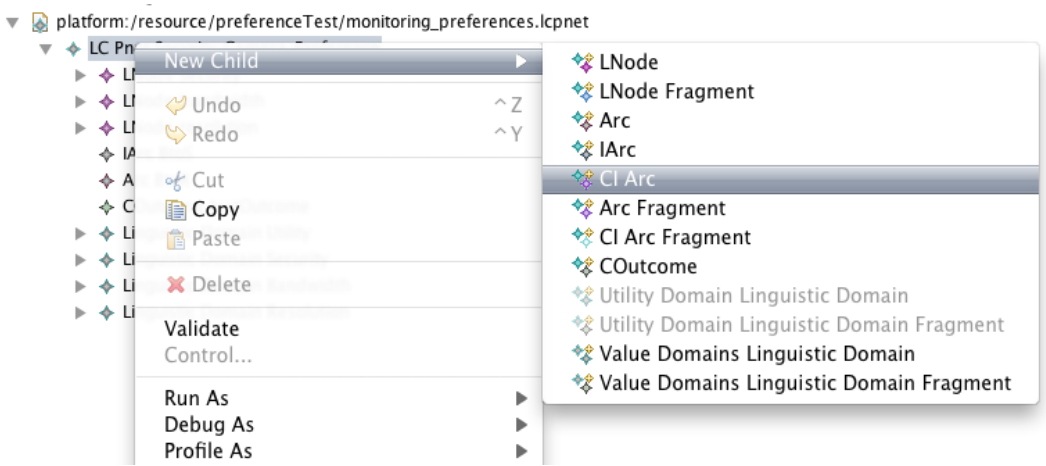

FIGURE 9.7 – Editeur de LCP-nets : ajout d'un ci-arc.

L'éditeur graphique mis au point dans ce canevas permet une définition plus aisée (et non programmatique) des préférences LCP-net et de leurs fragments, cette définition s'effectuant sous une forme arborescente.

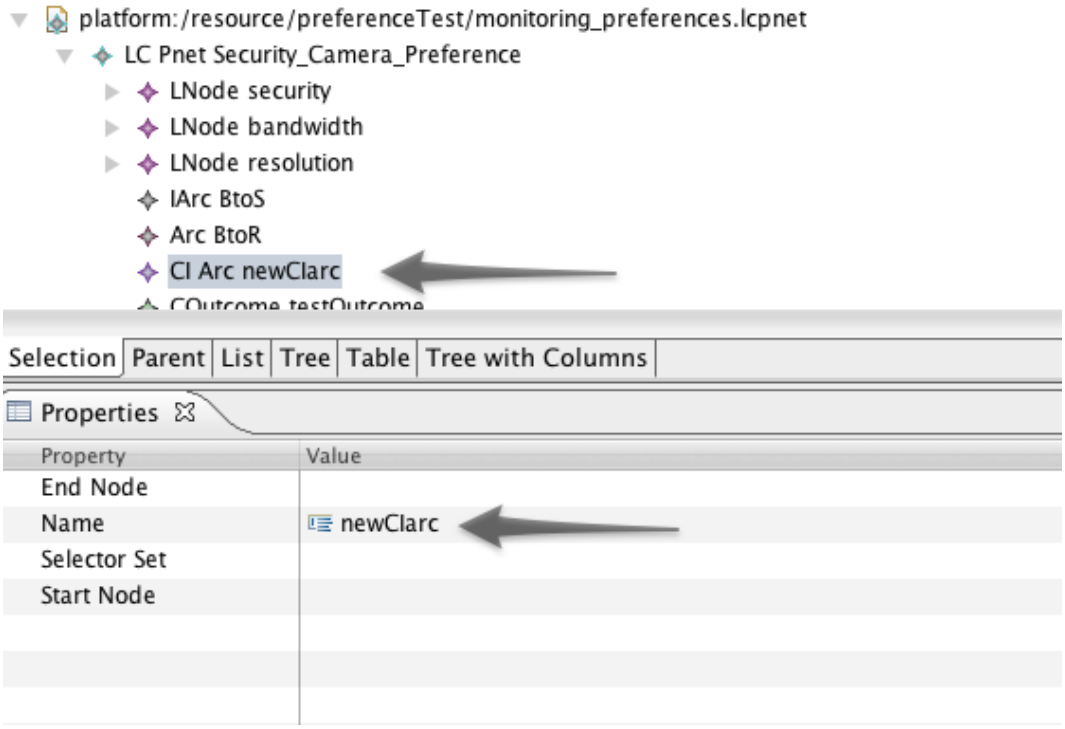

FIGURE 9.8 – Editeur de LCP-nets : modification des propriétés d'un ci-arc.

Cet éditeur a été créé sur la base de l'API LCP-net sous-jacente, il permet donc d'obtenir une sérialisation des modèles sous une forme XML facilement échangeable, à plus forte raison dans le contexte SSOA de ces travaux, sans aucun effort supplémentaire de la part de l'utilisateur.

Dans les copies d'écran suivantes, on illustre les différentes étapes menant à l'ajout et la modification d'un ci-arc au sein d'un LCP-net existant. Sur la figure 9.6, on a préalablement ouvert un fichier de fragment "modificator.fragment" sélectionné dans le panneau de gauche qui présente la liste des préférences accessibles sur le disque. L'éditeur de LCP-net est donc ouvert et présente de manière unifiée le fragment ainsi que son LCP-net cible.

Sur la figure 9.7 on illustre l'ajout d'un ci-arc à ce LCP-net cible, directement modifiable par un menu contextuel.

Ensuite, sur les figures 9.8 et 9.9, on procède à la modification des propriétés de ce nouvel arc. Dans le premier cas on modifie son nom, dans le deuxième, l'outil nous propose automatiquement la liste des nœuds cibles à utiliser pour compléter la définition du ci-arc.

Une évolution possible et souhaitée de cet éditeur va consister à bâtir, au dessus de l'existant grâce au canevas GMF ("Graphical Modeling Framework") d'Eclipse, un éditeur dont la représentation des LCP-nets soit au plus proche de celle des modèles présentés précédemment dans ce manuscrit : c'est-à-dire de manière à disposer de nœuds sous forme de cercles, d'arcs sous forme de flèches, etc.

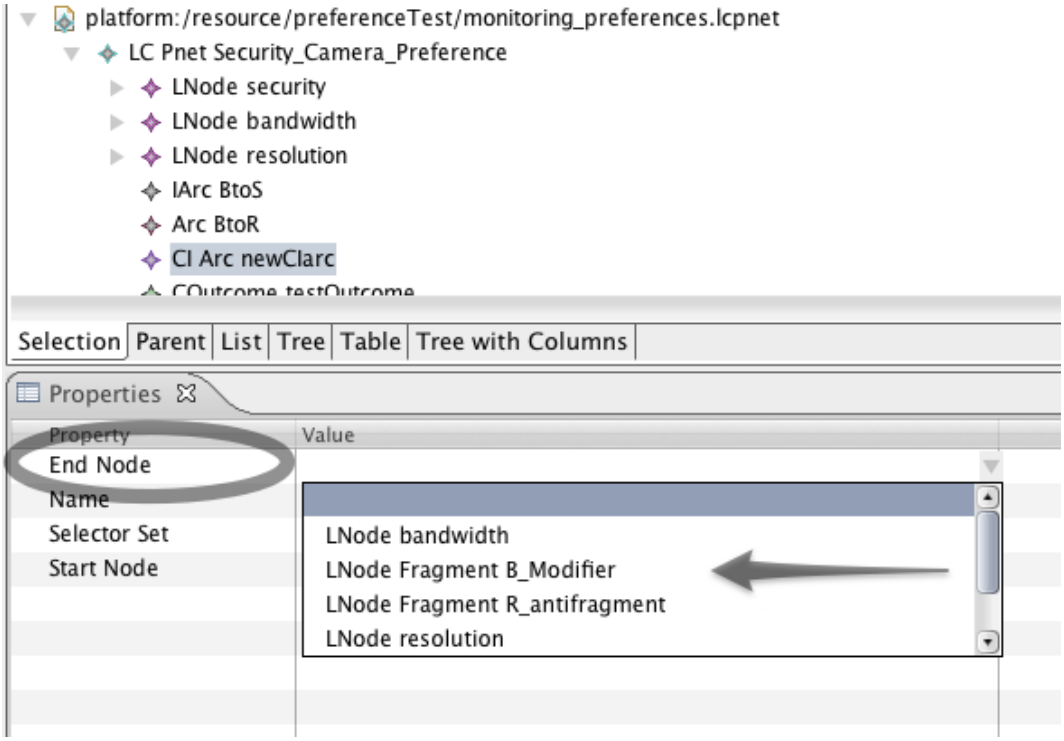

FIGURE 9.9 – Editeur de LCP-nets : accès à la liste des nœuds disponibles.

Procédure d'application des fragments

Du point de vue de l'utilisateur de la machine virtuelle LCP-net, l'application d'un fragment sur un LCP-net consiste simplement à appeler la primitive *getCombinedLCPnet(LCPnetFragment)*. Le lien entre le fragment et son LCP-net cible ayant été effectué lors de sa conception, c'est une information qu'il n'est plus nécessaire de préciser lors de l'application.

Du point de vue de l'implantation du canevas, cet appel de primitive est en fait redirigé vers la méthode d'instance *applyFragment(EObjet)* définie sur la classe *Fragment* dont hérite *LCPnetFragment*. Le *patron de conception "Chaîne de responsabilité"* a été choisi pour son implantation : il peut être utilisé *"dès lors qu'une information doit recevoir plusieurs traitements, ou juste être transmise entre différents objets"* et *"permet à un nombre quelconque de classes d'essayer de répondre à une requête sans connaître les possibilités des autres classes sur cette requête. Cela permet de diminuer le couplage entre objets. Le seul lien commun entre ces objets étant cette requête qui passe d'un objet à l'autre jusqu'à ce que l'un des objets puisse répondre." (Wikipedia)*

Cette approche est rendue possible, car tous les fragments héritent de la classe *Fragment* (cf. figure 9.1), une implantation de cette méthode spécifique à chaque type de fragment a donc été réalisée. Par ailleurs, de manière à ce que chaque sous-élément puisse s'appliquer correctement, la méthode *applyFragment(EObject)* dispose d'un argument correspondant au contexte d'application, qui va être transmis entre les différentes instances.

Ce principe est illustré par la figure 9.10 où il est possible de visualiser la *propagation en cascade* d'un appel par la VM (initié par l'utilisateur) à *applyFragment(null)* sur un fragment chargé en

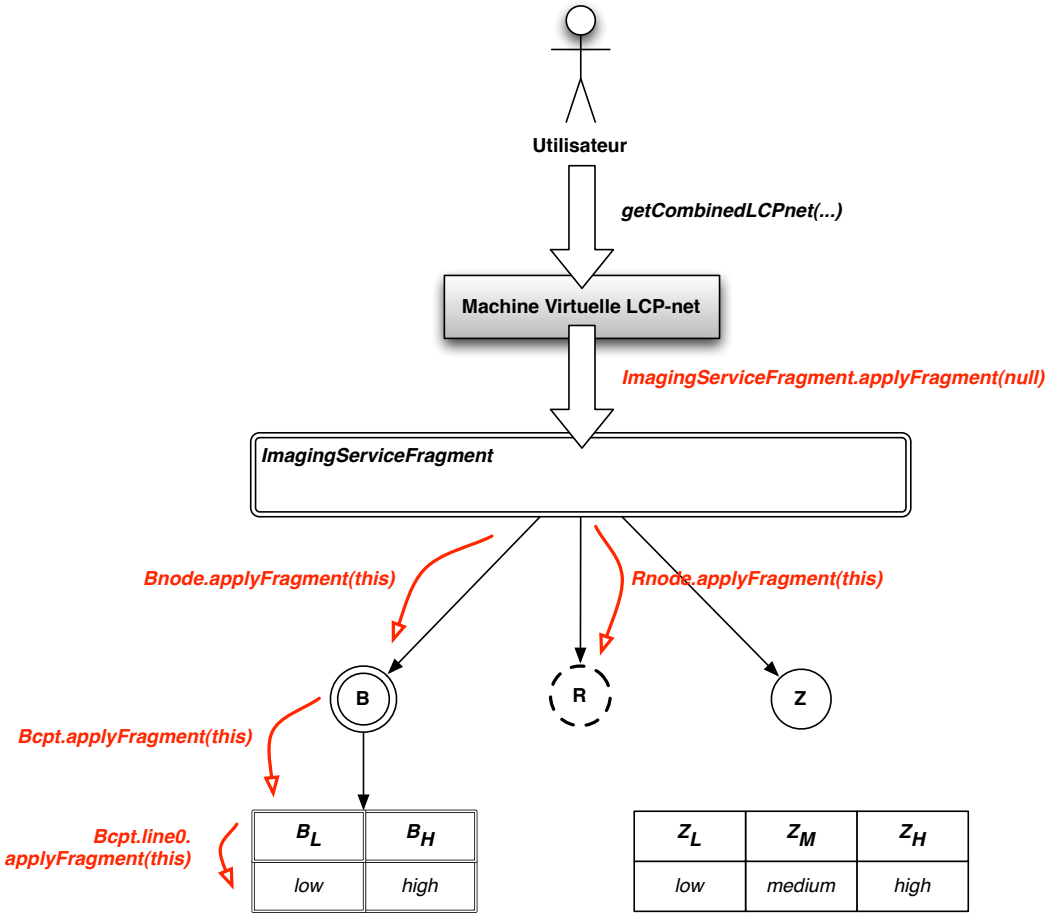

FIGURE 9.10 – Application d'un fragment de LCP-net en cascade.

mémoire selon le modèle de données objet, vers chacun de ses sous-fragments de nœuds, puis vers les tables, les lignes de tables, etc. A chacune de ces étapes, le fragment en question a la responsabilité de s'appliquer correctement et de transmettre le message vers ses sous-éléments.

En valeur de sortie de l'appel de méthode initial sur le fragment, le LCP-net résultant est obtenu. Il est totalement indépendant du LCP-net cible et du fragment.

9.2.3 Calculs et décisions sur LCP-nets

Le processus d'évaluation des préférences, à partir duquel la valeur d'utilité globale d'une affectation de valeurs aux nœuds d'un LCP-net est obtenue, a déjà été détaillé à la section 6.4.3 : il s'agit d'un problème de décision multi-critères, où chaque critère est une variable (donc un nœud) du réseau de préférences. Le but de cette section est de s'arrêter sur les points d'intérêt propres à l'implantation de ce processus.

Le point de départ du processus est incarné par un appel à la primitive *getUtility(LCPnet target, List names, List values)* de la machine virtuelle. Cet appel correspond à celui d'une méthode

d'instance éponyme sur le LCPnet *target* qui va calculer l'utilité globale de l'affectation en suivant l'algorithme suivant :

```
public double getUtility() throws LCPnetEvaluationException {
        // initialisation de l'utilité à 0 avant le calcul
        utility = 0;

        //1 - Conversion du réseau de préférences en graphe finalisé
        LCPnet net = (LCPnet) this.getPreferenceNetwork();
        Graph<LNode, Void> graph = net.getFinalizedGraph(this);

        //2 - Calcul des poids de chaque noeud en fonction de sa profondeur
        DepthBasedWeights.run(graph);

        //3 - Calcul de l'utilité globale à partir de chaque utilité locale
        for(Iterator<Vertex<LNode, Void>> iv=graph.getVertices(); iv.hasNext();) {
                Vertex<LNode, Void> vertex=iv.next();

                // 3.a - Calcul de l'utilité locale par un processus d'inférence floue
                LNode lnode = vertex.getVertex();
                double localUtility =  lnode.getUtility(this);

                // 3.b - Obtention du poids accordé au noeud
                Double localWeight = (Double) vertex.getUserFeature(DepthBasedWeights.
                    weightKey);

                // 3.c - Ajout de l'utilité locale pondérée de ce noeud à l'utilité globale
                utility += (localUtility * localWeight);
        }

        return utility;
}
```

La première étape de l'algorithme consiste à convertir le réseau de préférences en graphe finalisé (complètement orienté), le but étant de pouvoir calculer la profondeur de chaque nœud. Ce processus va consister à fixer l'orientation de chaque ci-arc, qui n'est pas connu à l'avance, en fonction des valeurs des nœuds : en effet, il s'agit du seul type d'arc dont la direction n'est pas connue au moment de la définition du LCP-net, leur orientation conditionnelle étant déterminée par les tables CIT.

Une fois le graphe finalisé, il est possible de calculer le poids respectif des nœuds lors de *la seconde étape*. Il s'agit d'un algorithme à part entière dont nous retraçons seulement ici les principaux aspects : il est effectué en deux passes, la première va consister à effectuer un parcours en profondeur du graphe finalisé ; la seconde va utiliser une fonction décroissante de distribution des poids f : *profondeur* $\rightarrow$ *poids* définie sur un domaine spécifique pour obtenir la pondération du nœud (cf. notion de fonction "BUM" à la section 8.3). Les poids ainsi obtenu sont ensuite normalisés pour totaliser 1, afin d'être correctement intégrés dans le calcul de moyenne pondérée effectué par cet algorithme.

La troisième et dernière étape consiste à mettre à exécution la moyenne pondérée des valeurs d'utilité locale à chaque nœud, de manière à obtenir l'utilité globale. On remarque que l'algorithme effectue un simple parcours de la liste des nœuds pour calculer chaque utilité locale "crisp" et l'ajouter à la valeur globale d'utilité en prenant en compte son poids relatif précédemment calculé.

Le calcul de l'utilité locale (étape 3.a) est effectué par un processus d'inférence floue dont une représentation visuelle est donnée à la section 6.4.3. Du point de vue de l'implantation, l'appel à

la méthode *getUtility* d'un nœud va déclencher sa traduction sous forme de Système d'Inférence Floue ("Fuzzy Inference System" ou FIS) [Jang, 1993] décrit ici à l'aide du langage normatif FCL ("Fuzzy Control Language"), tel que défini par la spécification *IEC 61131-7* [IEC, 2001]. Cette étape de traduction est en majeure partie fondée sur le contenu de la table CPT rattachée à chaque nœud considéré et *permet d'obtenir un FIS par table, donc par nœud* (cf. codes 6.3, 6.4 et 6.5).

On utilise un *patron de conception "singleton"* pour optimiser l'évaluation des préférences : de cette manière on ne procède au calcul du FIS de chaque nœud qu'une seule et unique fois au cours du cycle de vie d'un LCP-net. Le même LCP-net pourrait en effet être évalué à de multiples reprises, avec des valeurs de variables différentes, on évite ainsi de passer par cette étape de traduction à chaque reprise.

Suite à la traduction du contenu d'une CPT sous forme de système d'inférence, ce dernier est injecté dans un *moteur d'inférence floue*, avec les valeurs des variables du problème. Le choix d'un moteur adapté est discuté dans la sous-section suivante. Cependant, le fait de reposer sur le langage FCL nous rend indépendant, dans une certaine mesure, du moteur utilisé. Ce dernier pourrait être remplacé, en fonction des contraintes du contexte d'utilisation du canevas ou des futurs développements, par un autre moteur implantant la spécification *IEC 61131-7*.

Par ailleurs, la valeur d'utilité globale que nous souhaitons obtenir étant précise, nous décidons de fixer à l'avance (comme nous le permet le langage FCL) certains paramètres de la défuzzification de l'utilité nécessairement effectuée en fin de raisonnement flou sur le FIS de chaque nœud :

- La méthode d'accumulation (paramètre *'ACCU'*), c'est-à-dire l'agrégation des résultats flous (sous forme de SEF) des différentes règles d'un même FIS, de manière à obtenir un SEF agrégé représentatif, est fixée au calcul du maximum (valeur *'MAX'*) des SEFs. C'est une méthode simple, qui ne va pas forcément donner le résultat le plus représentatif des entrées, mais qui est particulièrement rapide, ce qui est important dans notre contexte SSOA où l'inférence est effectuée en cours d'orchestration d'un processus métier.

- La méthode de défuzzification (paramètre *'METHOD'*) est fixée au calcul du centre de gravité (valeur *'COG'*) du SEF d'utilité obtenu. De même que pour l'accumulation, le choix a été ici de prendre une méthode simple mais rapide.

Choix d'un moteur d'inférence floue

Le processus d'inférence floue en lui-même repose sur l'utilisation d'un raisonneur logique externe (ou "moteur d'inférence") spécialisé dans la logique floue : *jFuzzyLogic*[16]. Plusieurs critères ont dirigé son choix comparativement à d'autres alternatives telles que *mbFuzzit*[17], *NRC FuzzyJ Toolkit*[18] ou *Free Fuzzy Logic Library*[19] :

- Il présente tout d'abord, et c'est un pré-requis dans le contexte de nos travaux, une licence LGPL compatible avec le reste du projet SemEUsE.

- Il s'agit d'un projet relativement récent comparativement à certaines de ses alternatives.

16. http ://jfuzzylogic.sourceforge.net
17. http ://mbfuzzit.sourceforge.net/en/mbfuzzit_software.html
18. http ://www.iit.nrc.ca/IR_public/fuzzy/fuzzyJDocs/index.html
19. http ://ffll.sourceforge.net/index.html

– Il offre de nombreuses possibilités de paramétrage sur le déroulement du processus d'inférence, par exemple en termes de défuzzification, qui peut être effectuée à partir des méthodes usuelles, telles que vue dans la section 3.1.1, ou par la définition d'opérateurs de défuzzification personnalisés.

– Il implante le langage FCL, et l'utilise pour la représentation des FIS.

Limitations de l'implantation actuelle

Cependant, l'utilisation de *JFuzzyLogic* entraîne aussi des limitations au niveau plus général du canevas de décision. En particulier, cette bibliothèque nous oblige à traiter les potentielles entrées floues du processus d'inférence comme des valeurs précises ("crisp"), ce qui induit un processus de défuzzification préalable à leur injection dans le moteur d'inférence. En effet, dans son implantation actuelle, *JFuzzyLogic* ne supporte que la définition de systèmes d'inférence floue disposant d'entrées et de sorties typées comme étant des réels, comme dans l'extrait de FIS au format FCL ci-dessous :

```
FUNCTION_BLOCK fbName

VAR_INPUT
        B : REAL;
        R : REAL;
END_VAR

VAR_OUTPUT
        Utility : REAL;
END_VAR

(...)
```

Dans l'exemple donné à la section 6.3.2 sur la sélection d'un drone d'imagerie, les variables d'entrée (R, B, S) et de sortie ($Utility$) des trois FIS obtenus à partir du modèle de préférences (cf. figure 6.1 et codes 6.3, 6.4 et 6.5) doivent donc être fixées comme étant de type réel. Si une valeur floue de résolution, de bande passante ou de sécurité est soumise lors de l'évaluation, elle devra donc être préalablement *defuzzifiée*. A l'inverse, si la valeur d'utilité en sortie du processus d'inférence est souhaitée sous forme linguistique, il faudrait avoir recours à un processus inverse de fuzzification en rapport avec le partitionnement flou déjà existant du domaine de l'utilité.

Ce type de limitations pourrait être en partie levé par la mise au point, à terme, de notre propre moteur d'inférence flou reposant, par exemple, sur l'utilisation des 2-tuples flous pour mener à bien ses calculs sans la perte d'information induite par les étapes de fuzzification/défuzzification successives [Herrera et Martínez, 2000].

9.2.4 Retour sur les principaux choix d'implantation

Cette section présente un bref rappel des différents choix technologiques et techniques effectués pour la réalisation du canevas de modélisation et décision sur LCP-nets, leurs motivations ont été distillées au cours des précédentes sections :

- Réalisation effectuée en totalité en Java sous Eclipse, avec la mise en œuvre de plusieurs patrons de conception usuels en Génie Logiciel, de manière à faciliter le développement ou optimiser le code : *encapsulation des données*, *singleton*, *chaîne de responsabilité* etc. Ce dernier permet la réalisation de l'application d'un fragment par répartition de la charge d'application à tous ses sous-éléments.
- Utilisation du méta-modèle EMF et de son canevas pour la modélisation des LCP-nets et la génération de leur modèle de données. Un modèle générique de tous les *CP-nets a en fait été établi.
- Sérialisation des réseaux de préférences sous forme XML, traitée par les bibliothèques de support EMF.
- Réalisation du développement du canevas en totale indépendance du contexte SSOA et de SemEUsE, bien qu'il réponde aux contraintes que ces derniers imposent. C'est la garantie de la généricité du canevas et de son potentiel dans d'autres contextes de décision multi-critères.
- Intégration d'un moteur d'inférence en logique floue pré-existant, *JFuzzyLogic*, en fixant à l'avance certains de ses paramètres pour le raisonnement. On reste cependant, à un certain degré, indépendant du moteur utilisé. Ce dernier pourrait être amené à changer en fonction des contraintes du contexte d'utilisation.

9.3 Canevas de liaison tardive de services Web

Nos efforts pour la mise en œuvre de ce canevas ont porté sur l'intégration du processus de décision sur LCP-nets (cf. section 9.2), effectué ici sur la base des valeurs de QoS obtenues par la supervision de services *Web*, lors de leur liaison tardive à un processus métier *BPEL* (cf. section 5.3). Pour permettre cette intégration, en fonction des contraintes propres à SemEUsE et aux SSOA, différents choix techniques ont été effectués, ils seront présentés au cours de cette section.

9.3.1 Contexte technique et architectural

La liaison tardive est effectuée au cours de l'orchestration de services. Pour réaliser cette orchestration dans le cadre du projet SemEUsE, le moteur Orchestra a été choisi. Il a été brièvement présenté dans le contexte technique de ce manuscrit (cf. section 2.1.4). Sa particularité est de proposer une implantation de la spécification WS-BPEL 2.0 et de son concept d'extension d'activités.

Le canevas de liaison tardive est représenté sous la forme d'un composant dans l'architecture générale de SemEUsE, tel qu'illustré par la figure 9.11. On remarque qu'il se trouve à la jonction de la composition et de la supervision de services. Sur cette figure, on met en correspondance les étapes logiques (déjà présentées à la figure 5.4) qui permettent d'arriver à la décision de liaison tardive d'un service (sur la droite de la figure), avec les composants de l'architecture générale (sur sa gauche).

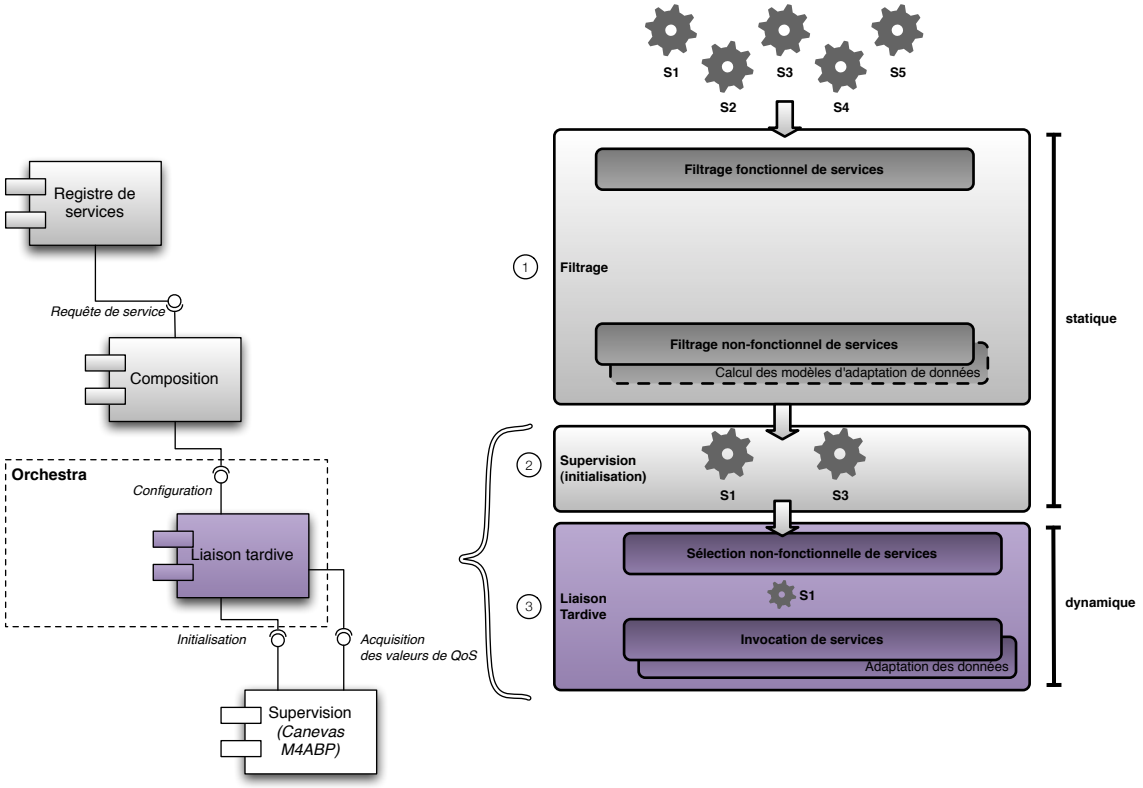

FIGURE 9.11 – Positionnement du composant de liaison tardive dans l'architecture SemEUsE.

9.3.2 Implantation de l'activité *lateBindingInvoke* dans Orchestra

Dans le chapitre 7 nous avons introduit l'activité *lateBindingInvoke* de liaison tardive de services, nous précisions là les subtilités propres à son implantation dans Orchestra.

Comme il a été discuté à la section 5.4.2, plusieurs options s'offrent à nous quant à la répartition temporelle des tâches préalables à la liaison tardive, que sont le *filtrage des services* et le *calcul des modèles d'adaptation de données*. De manière à limiter leur impact sur le processus d'orchestration de services, nous préconisons l'approche illustrée par la figure 9.12 :

- *Le filtrage des services* de l'annuaire est effectué au chargement du processus, lors de la composition. Dans le contexte SSOA, la recherche sur l'annuaire va être effectuée en fonction de la sémantique métier rattachée aux offres et demandes de services SAWSDL *via* des concepts ontologiques. Ce raisonnement ontologique, d'une durée non-négligeable, doit effectivement être effectué avant l'orchestration.
- *Le calcul des modèles d'adaptation de données*, doit être effectué dès les services connus, donc après le filtrage. Attendre le moment (plus tardif) de l'invocation d'un service pourrait, là aussi, avoir des répercussions importantes sur la réactivité globale de l'orchestration. En effet, calculer une adaptation entre deux types de données annotés sémantiquement est un problème non trivial.

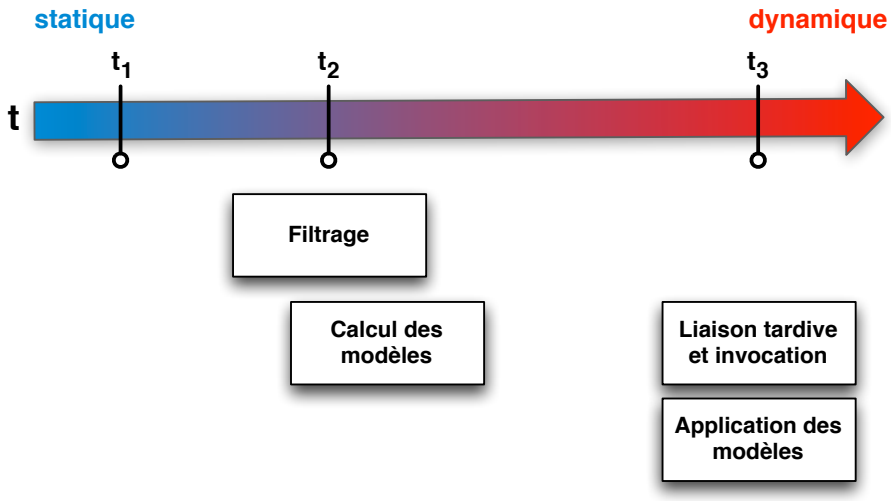

FIGURE 9.12 – Répartition temporelle du filtrage, de l'adaptation et de l'invocation de services dans l'implantation.

Comme indiqué par la figure 9.11, le filtrage est la responsabilité du composant de composition de services. En conséquence de quoi, *l'activité de liaison tardive doit être en mesure d'intégrer le résultat de ce filtrage.*

Par ailleurs, de manière à obtenir des valeurs de QoS courantes des services, le composant de liaison tardive est client d'un canevas de supervision. *Ce canevas doit être initialisé le plus tôt possible* avec la liste des services filtrés, pour chaque site d'appel de service d'un processus. Cette initialisation précoce permet de mettre en place la remontée des valeurs de QoS vers l'orchestration, de manière à pouvoir en disposer immédiatement au moment de la sélection d'un service.

Nous décidons donc d'implanter la liaison tardive non pas sous la forme d'une seule activité étendue WS-BPEL 2.0 *lateBindingInvoke*, mais de deux, aux responsabilités distinctes :

1. **une activité lateBindingConfigure :** placée au tout début du processus métier de manière à être évaluée avant le code métier. Elle reprend des éléments de la syntaxe initiale (cf. code 7.3), notamment la liste des services filtrés *<semeuse :candidateServices>...</semeuse :candidateServices>* fourni par la composition. Son implantation effectue l'initialisation du canevas de supervision grâce aux informations contenues dans sa représentation XML : les *EPR* ("End-Point Reference") des services Web, leur *portType*, et informations connexes sur les dimensions de QoS à superviser obtenues dans les fichiers *WS-Agreement* liés. Ces "agreements" sont obtenus à l'issue d'une phase de *négociation des contrats non-fonctionnels* lors de la composition de services.

   Un exemple de la syntaxe XML de cette activité étendue dans un processus WS-BPEL 2.0 est donnée ci-dessous :

```
<extensionActivity>
        <semeuse:lateBindingConfigure invocationID="invocation1" preference="tests/
            semeuse/activities/monitoring/monitoring_preferences.lcpnet">
```

```
                <semeuse:candidateServices>
                        <semeuse:service EPR="service1" portType="service1PortType"
                            operation="getVideo" contract="negociated_wsag_camera1.xml"
                            />
                        <semeuse:service EPR="service2" portType="service2PortType"
                            operation="getVideo" contract="negociated_wsag_camera2.xml"
                            />
                        <semeuse:service EPR="service3" portType="service2PortType"
                            operation="getVideo" contract="negociated_wsag_camera3.xml"
                            />
                        <semeuse:service EPR="service4" portType="service2PortType"
                            operation="getVideo" contract="negociated_wsag_camera4.xml"
                            />
                </semeuse:candidateServices>
        </semeuse:lateBindingConfigure>
</extensionActivity>
```

2. **une activité lateBindingInvoke** qui est liée à une activité *lateBindingConfigure* jumelle, par le biais d'un identifiant unique *invocationID* utilisé par les deux activités. Ces identifiants sont générés et positionnés automatiquement lors de la composition d'un processus métier abstrait où les services Web n'ont pas encore été filtrés, vers un processus BPEL disposant de ces activités étendues.

   Un exemple de la syntaxe XML de cette activité étendue dans BPEL est donné ci-dessous :

```
<extensionActivity>
        <semeuse:lateBindingInvoke invocationID="invocation1" inputVariable="invoke_in"
             outputVariable="invoke_out" preference="preferences.lcpnet"/>
</extensionActivity>
```

   Cette activité a la responsabilité d'effectuer la sélection effective d'un service, sur la base de sa QoS courante et des préférences LCP-net en vigueur, en vue de son invocation.

A la lumière de ces développements, on précise la topologie du canevas de liaison tardive dans SemEUsE, tel qu'il avait été décrit par la figure 9.11 : les interfaces avec les autres composants de l'architecture globale sont maintenant réparties entre les implantations distinctes des deux activités *lateBindingConfigure* et *lateBindingInvoke* (cf. figure 9.13). La structure interne du composant de supervision a aussi été exposée et sera détaillée dans la section 9.3.3. On introduit, par ailleurs, une interface avec un composant de *reconfiguration* : il s'agit ici de déclencher et de sous-traiter une recomposition du processus métier, si la liaison tardive n'est plus à même de se réaliser correctement, par exemple par manque de services disponibles. Ce type de reconfiguration est à rapprocher des travaux sur le *"re-binding"* de services évoqués dans notre état de l'art.

Ces deux activités sont mises en œuvre dans un processus BPEL d'exemple (cf. code 9.1) dont la partie métier n'est constituée que d'une simple invocation tardive. On remarque qu'elles sont complétées par une activité propre au composant de supervision *M4ABP* : pour déclencher la supervision en début de processus, dès sa configuration effectuée (*<semeuse :monitoring state="start" />*), puis pour l'arrêter à sa toute fin (*<semeuse :monitoring state="stop" />*).

Les interfaces de la liaison tardive, notamment avec le composant de supervision, ayant été précisées, il est possible maintenant de s'intéresser à l'implantation dans Orchestra des deux activités :

```
<process name="semeuse">
        (...)
        <partnerLinks>
                <!-- The 'client' role represents the requester of this service. -->
                <partnerLink name="client" partnerLinkType="tns:semeuse" myRole="semeuseProvider" />
        </partnerLinks>

        <variables>
                <!-- Reference to the message passed as input during initiation -->
                <variable name="input" messageType="tns:semeuseRequestMessage"/>

                <!-- Reference to the message that will be returned to the requester -->
                <variable name="output" messageType="tns:semeuseResponseMessage"/>

                <variable name="invoke_in" messageType="tns:invoke_request"/>
                <variable name="invoke_out" messageType="tns:invoke_response"/>
        </variables>

        <sequence name="main">

                <extensionActivity>
                        <semeuse:lateBindingConfigure invocationID="invocation1" preference="preferences.lcpnet">
                                <semeuse:candidateServices>
                                        (...)
                                </semeuse:candidateServices>
                        </semeuse:lateBindingConfigure>
                </extensionActivity>

                <extensionActivity>
                        <semeuse:monitoring state="start" />
                </extensionActivity>

                <!-- Receive input from requester.-->
                <receive name="receiveInput" partnerLink="client" portType="tns:semeuse" operation="process" variable="
                     input" createInstance="yes"/>

                <assign>
                        <copy>
                                <from>$input.payload/tns:input</from>
                                <to>$invoke_in.payload</to>
                        </copy>
                </assign>

                <sequence name="invocation">
                        <extensionActivity>
                                <semeuse:lateBindingInvoke invocationID="invocation1" inputVariable="invoke_in"
                                     outputVariable="invoke_out" preference="preferences.lcpnet"/>
                        </extensionActivity>

                        <assign>
                                <copy>
                                        <from>$invoke_out.payload</from>
                                        <to>$output.payload/tns:result</to>
                                </copy>
                        </assign>
                </sequence>

                <!-- Generate reply to synchronous request -->
                <reply name="replyOutput" partnerLink="client" portType="tns:semeuse" operation="process" variable="output"
                      />

                <extensionActivity>
                        <semeuse:monitoring state="stop" />
                </extensionActivity>
        </sequence>
</process>
```

Code 9.1 – Exemple de processus BPEL avec liaison tardive.

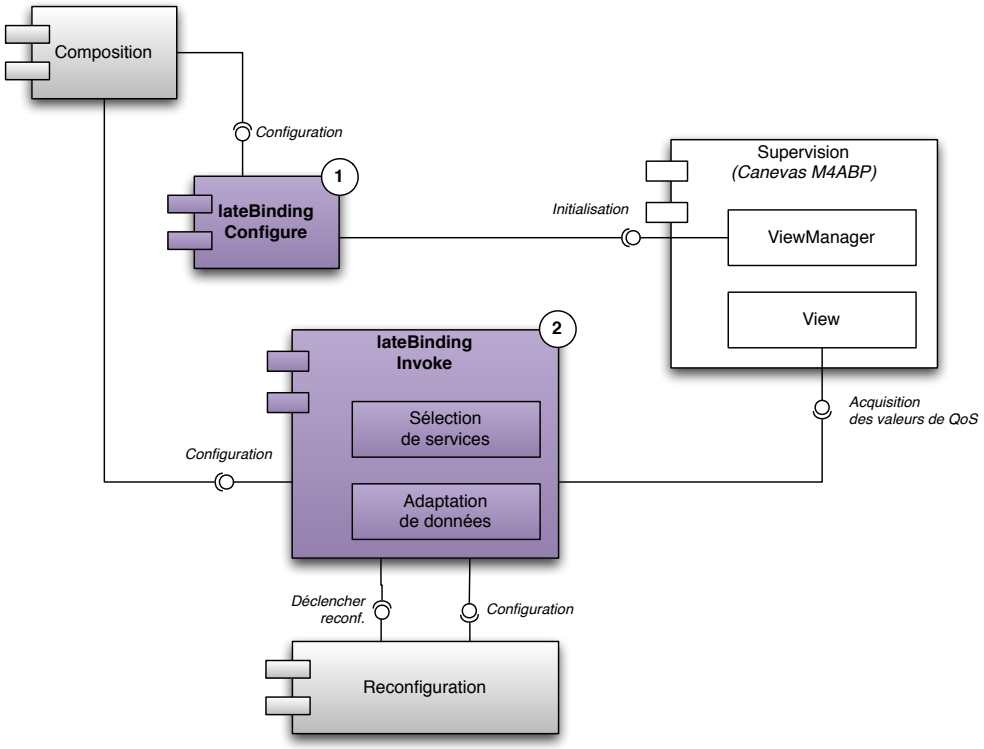

FIGURE 9.13 – *lateBindingConfigure* et *lateBindingInvoke* dans l'architecture SemEUsE.

– En tant qu'activités étendues, elles sont mises en œuvre sous formes de classes, héritant de *org.ow2.orchestra.definition.activity.AutomaticActivity* (cf. figure 9.14).

– Dans chacune de ces classes, la logique métier est définie dans une méthode *void executeBusinessLogic(BpelExecution execution)*.

– Pour faire le pont avec la sémantique opérationnelle de l'intégration des *HL*-LCP-nets et de leurs fragments dans les processus métiers (cf. section 7.4), l'environnement d'exécution ρ ainsi que les autres éléments de contexte γ sont ici représentés par l'argument *BpelExecution execution* de la précédente méthode.

Une vue simplifiée et commentée de l'implantation Java de l'activité *lateBindingConfigure* dans Orchestra est donnée par le fragment de code 9.2. On remarque que la liste des services obtenus par la composition, telle que visible dans le représentation XML de cette activité, est stockée dans un registre *Registry* commun aux deux activités, de manière à ce que *lateBindingInvoke* puisse y accéder *a posteriori* (il s'agit là aussi d'un élément d'environnement couvert par ρ et γ). Les paramètres propres à chaque service candidat de la liste XML *<semeuse :candidateServices>...</semeuse :candidateServices>* sont fournis au canevas M4ABP pour sa configuration. Cette dernière se termine en fixant (uniquement dans le code à l'heure actuelle) la *QoI* (*"Quality of Information"*), c'est-à-dire les besoins en termes de qualité sur la supervision elle-même : il s'agit ici de garantir dans un premier temps un certain niveau de cohérence temporelle entre les mesures effectuées par *M4ABP*.

```
public class LateBindingConfigureActivity extends AutomaticActivity {
        private String invocationID;
        private ArrayList<CandidateService> services = new ArrayList<CandidateService>();
        private String preference;

        protected void executeBusinessLogic(BpelExecution execution) {
                //1 - Enregistre la liste des services pour l'invocation tardive
                Registry.addCandidateServices(invocationID, services);

                //2 - Chargement du modèle de préférences
                LCPnet preferenceModel = LCPnetImpl.load(preference);

                //3 - Configuration de M4ABP (vie une Vue)
                ViewManager vm = fr.orange.monitoring.ViewManager.getViewManager(execution.getProcessInstanceUUID().
                     toString());
                ViewFactory factory = ViewFactory.eINSTANCE;
                DescriptionType viewType = factory.createDescriptionType();
                (...)

                List<DimensionType> triplets = viewType.getDimension();
                //3.a - Initialisation de la supervision pour CHAQUE service setup
                for (CandidateService candidateService : services) {
                        String contract = candidateService.getContract();
                        String operation = candidateService.getOperation();

                        //Pour CHAQUE propriété de QoS
                        //Les noms des propriétés de QoS sont ceux utilisés dans le LCP-net
                        EList<String> properties = preferenceModel.getNodesNames();
                        for (String property : properties) {
                                DimensionType dimension = factory.createDimensionType();

                                dimension.setWsAgreementURL(contract);
                                dimension.setServiceName(operation);
                                dimension.setProperty(property);

                                triplets.add(dimension);
                        }
                }

                //3.b - Configuration de la QoI
                Properties props = factory.createProperties();
                viewType.setProperties(props);

                Property prop = factory.createProperty();
                prop.setKey("coherency");
                prop.setValue("1000");
                props.getProperty().add(prop);
                (...)
        }
(...)
```

Code 9.2 – Implantation de l'activité lateBindingConfigure.

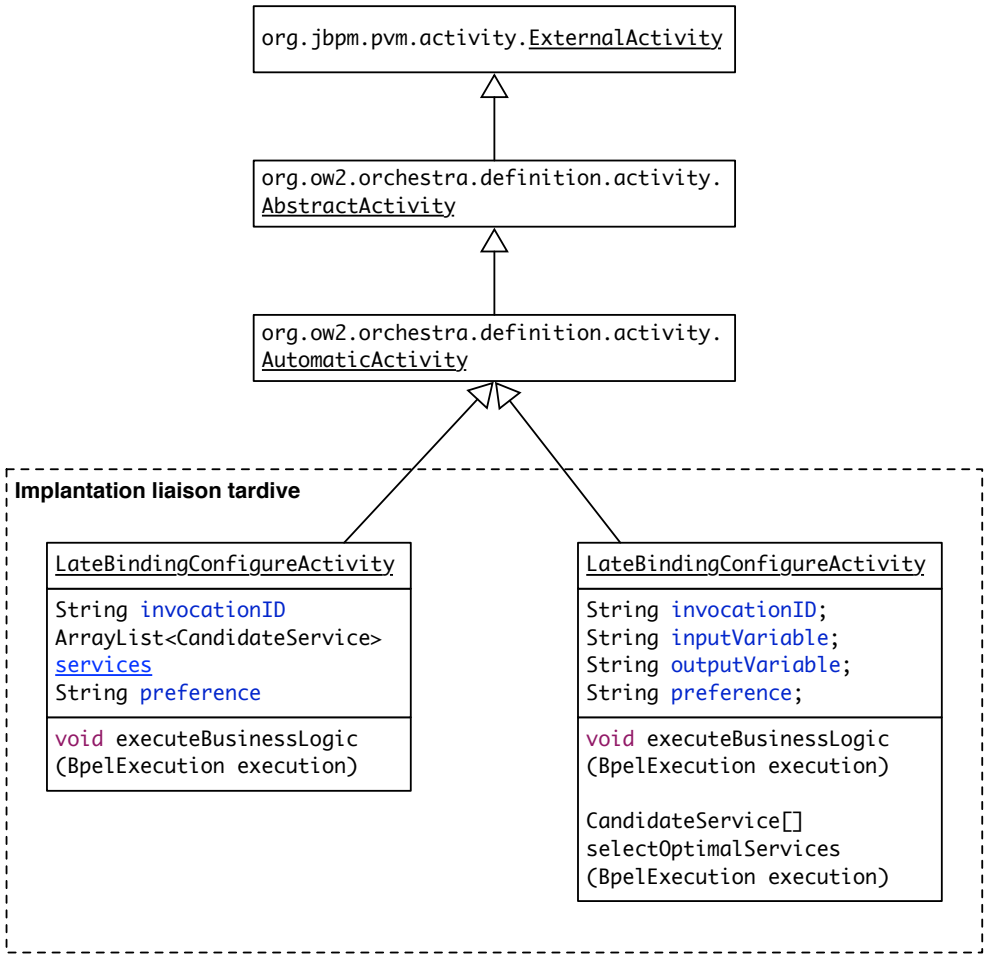

FIGURE 9.14 – Diagramme de classes UML des activités *lateBindingConfigure* et *lateBindingInvoke* dans Orchestra.

L'implantation de l'activité *lateBindingInvoke* dans Orchestra est quant à elle donnée par le fragment de code 9.3. On remarque ici que l'essentiel du travail de sélection est effectué par une méthode *selectOptimalServices(...)* dédiée, qui peut retourner une liste de services optimaux dans l'éventualité (peu probable) où plusieurs services exhiberaient une utilité globale strictement similaire : chacune des étapes clés du processus de sélection est commentée directement dans l'extrait de code. Parmi ces étapes, il est important de relever celle concernant l'accès à une abstraction de supervision *View* qui constitue l'interface unifiée par laquelle les valeurs courantes de *toutes* les propriétés de QoS de *tous* les services supervisés *pour cette invocation* (identifiant *invocationID*) sont obtenues.

```
public class LateBindingInvokeActivity extends AutomaticActivity {
        private String invocationID;
        private String inputVariable;
        private String outputVariable;
        private String preference;

        protected void executeBusinessLogic(BpelExecution execution) {
                CandidateService[] optimalServices = this.selectOptimalServices(execution);
                this.invokeOptimalService(optimalServices, execution);
        }

        private CandidateService[] selectOptimalServices(BpelExecution execution) throws Exception{
                //1 - On accède à la vue dédiée à cette invocation (grâce à invocationID)
                ViewManager vm = ViewManager.getViewManager(execution.getProcessInstanceUUID().toString());
                View view = vm.getView(invocationID);

                //2 - Chargement du modèle de préférences
                LCPnet preferenceModel = LCPnetImpl.load(preference);

                //3 - Calcul de l'utilité globale de chaque service candidat et selection
                ArrayList<CandidateService> optimalServices = new ArrayList<CandidateService>();
                double bestUtilityValue = -1;

                //3.1 - On accède à la liste des services précédemment stockée
                ArrayList<CandidateService> services = Registry.getCandidateServices(invocationID);
                for (CandidateService candidateService : services) {
                        //3.2 - On accède aux QoS courantes via la vue fournie par M4ABP

                        //Les noms des propriétés de QoS sont ceux utilisés dans le LCP-net
                        EList<String> properties = preferenceModel.getNodesNames()
                        BasicEList<Double> propertiesValues = new BasicEList<Double>();
                        for (String property : properties) {
                                DimensionType dimension = factory.createDimensionType();
                                dimension.setProperty(property);

                                //Requête de la QoS courante envoyée à M4ABP
                                QoSData data = view.getData(dimension);
                                propertiesValues.add(new Double((String)data.getQoSValue()));
                        }

                        //3.3 - Calcul de l'utilité globale d'UN service en fonction des ses valeurs courantes de QoS
                        double candidateServiceUtility = preferenceModel.getUtility(properties, propertiesValues);

                        if(candidateServiceUtility > bestUtilityValue) {
                                //C'est le service sélectionné (pour l'instant)
                                optimalServices = new ArrayList<CandidateService>();
                                optimalServices.add(candidateService);
                                bestUtilityValue = candidateServiceUtility;
                        } else if(candidateServiceUtility == bestUtilityValue) {
                                //Il y a plus d'un service sélectionné (utilités globales strictement identiques)
                                optimalServices.add(candidateService);
                        }
                }
                return optimalServices.toArray(new CandidateService[]{});
        }
(...)
```

Code 9.3 – Implantation de l'activité lateBindingInvoke.

9.3.3 Configuration et intégration de la supervision de services

Dans cette section, nous apportons plusieurs précisions sur le mécanisme de configuration du canevas de supervision ainsi que sur le mode d'intégration des valeurs de supervision lors de la sélection de services en liaison tardive.

Configuration de la supervision

La configuration du canevas de supervision par l'activité *lateBindingConfigure* dépend, en majeure partie, des informations contenues dans le WS-agreement *négocié* rattaché à chaque service de la liste des candidats. L'activité elle-même ne va pas interpréter ces données, mais les transmettre directement au canevas. Un WS-Agreement négocié d'exemple est présenté séparé en deux parties par les extraits de code 9.4 et 9.5. Ils font apparaître :

- en première partie les propriétés *<wsag :ServiceProperties ...>* requises (par le processus) et offertes (par le service considéré) de QoS (*Resolution*, *Bandwidth*, *BandePassante*),
- suivies des garanties (contraintes) portant sur ces propriétés *<wsag :GuaranteeTerm ...>*. Elles sont exprimées dans une extension du langage QML [Frølund et Koistinen, 1998a] mise au point dans le cadre de SemEUsE : *WS-QML* ; de manière à être intégrées dans WS-Agreement. En l'occurrence, il s'agit de bornes numériques à ne pas dépasser : pour qu'un service soit intégré lors de la composition dans la liste des candidats, ses garanties en termes de QoS doivent être identiques ou plus strictes que celles requises par le processus métier.

Dans l'implantation de l'activité *lateBindingInvoke*, tout comme dans celle de *lateBindingConfigure*, il est important de noter que les noms des propriétés de QoS utilisés, pour mener à bien les requêtes de supervision, sont ceux utilisés dans le modèle de préférences LCP-net. Il s'agit de la conséquence d'une contrainte de modélisation préalable, où les préférences sont censées, pour des raisons de cohérence, utiliser des noms identiques aux propriétés fixées dans le WS-agreement *requis* par le processus métier.

La conséquence de cette approche est que chaque WS-agreement *négocié* (utilisé pour la configuration de *M4ABP*) entre un fournisseur et un consommateur de service, doit contenir les informations de correspondance entre la terminologie utilisée par les services pour décrire leurs propriétés et celle du processus, pas forcement identique. Cette correspondance et ensuite gérée de manière transparente, pour l'activité *lateBindingInvoke* cliente, par le canevas de supervision. Un exemple de rétention de cette correspondance est clairement visible dans la première partie du WS-agreement négocié (cf. code 9.4) : une dimension requise est nommée *Bandwidth* alors qu'elle est offerte sous le nom de *BandePassante*. Toutes deux sont cependant reliées au même concept ontologique *http ://www.nca.or.kr/2006/wsqdl/QualityFactor/EvalFactor/Bandwidth*. Ces notions sont résumées dans la figure 9.15.

```
<wsag:Agreement AgreementId="myFirstSSOAAgreement">
  <wsag:AgreementContext>
      ...
  </wsag:AgreementContext>
  <wsag:Terms>
       <wsag:all>
         ...
         <wsag:ServiceProperties wsag:Name="techniqueRequis" wsag:ServiceName="requis">
              <wsag:VariableSet>
                <wsag:Variable wsag:Name="Resolution" wsag:Metric="http://www.w3.org/2007/ont/unit#megapixels">
                      <wsag:Location>
                        <ws-qml:QoS-dimension
                              ws-qml:semantic-concept="http://www.photo.org/QualityFactor/MeasureFactor/Resolution"/>
                      </wsag:Location>
                </wsag:Variable>
                <wsag:Variable wsag:Name="Bandwidth" wsag:Metric="http://www.w3.org/2007/ont/unit#megabitsps">
                      <wsag:Location>
                        <ws-qml:QoS-dimension
                              ws-qml:semantic-concept="http://www.nca.or.kr/2006/wsqdl/QualityFactor/EvalFactor/Bandwidth
                                    "/>
                      </wsag:Location>
                </wsag:Variable>
              </wsag:VariableSet>
         </wsag:ServiceProperties>

         <wsag:ServiceProperties wsag:Name="techniqueOffert" wsag:ServiceName="offert">
              <wsag:VariableSet>
                <wsag:Variable wsag:Name="Resolution" wsag:Metric="http://www.w3.org/2007/ont/unit#megapixels">
                      <wsag:Location xsi:type="semeuse:WSBinding" mode="pull" type="ws"
                                          epr="http://localhost:8080/monitoring_firefighting/services/Camera1"
                                          operation="getResolution">
                        <ws-qml:QoS-dimension
                              ws-qml:semantic-concept="http://www.photo.org/QualityFactor/MeasureFactor/Resolution"/>
                        <ws-qml:semantic-match requiredTargetName="Resolution"/>
                      </wsag:Location>
                </wsag:Variable>
                <wsag:Variable wsag:Name="BandePassante" wsag:Metric="http://www.w3.org/2007/ont/unit#megabytesps">
                      <wsag:Location xsi:type="semeuse:WSBinding" mode="pull" type="ws"
                                          epr="http://localhost:8080/monitoring_firefighting/services/Camera1"
                                          operation="getBandwidth">
                        <ws-qml:QoS-dimension
                              ws-qml:semantic-concept="http://www.nca.or.kr/2006/wsqdl/QualityFactor/EvalFactor/Bandwidth
                                    "/>
                        <ws-qml:semantic-match requiredTargetName="BandePassante">
                              <ws-qml:conversion-function sourceMetric="http://www.w3.org/2007/ont/unit#megabytesps"
                                                                                    targetMetric="http://www.w3.org
                                                                                          /2007/ont/unit#megabitsps">
                                <ws-qml:function>
                                      <ws-qml:param name="MBs"/>
                                      <ws-qml:expression>
                                        <ws-qml:binary-expression operator="*">
                                              <ws-qml:integer-constant value="8"/>
                                              <ws-qml:variable name="MBs"/>
                                        <ws-qml:binary-expression>
                                      </ws-qml:expression>
                                </ws-qml:function>
                              </ws-qml:conversion-function>
                        </ws-qml:semantic-match>
                      </wsag:Location>
                </wsag:Variable>
              </wsag:VariableSet>
         </wsag:ServiceProperties>

         ...
```

Code 9.4 – Extrait de WS-Agreement négocié : description des QoS requises et offertes.

```
        ...
        <wsag:GuaranteeTerm wsag:name="garantieRequise" wsag:Obligated="ServiceConsumer">
              <wsag:ServiceScope>...</wsag:ServiceScope>
              <wsag:ServiceLevelObjective>
                <wsag:KPITarget>
                      <wsag:KPIName>Resolution</wsag:KPIName>
                      <wsag:Target>
                        <ws-qml:constraint ws-qml:operator=">">
                              <ws-qml:value>
                                2
                                <ws-qml:unit ws-qml:name="megapixels"
                                      ws-qml:semantic-concept="http://www.w3.org/2007/ont/unit#megapixels"/>
                                <ws-qml:type ws-qml:name="decimal"
                                      ws-qml:semantic-concept="http://www.w3c.org/2001/XMLSchema:integer"/>
                              </ws-qml:value>
                        </ws-qml:constraint>
                      </wsag:Target>
                </wsag:KPITarget>
              </wsag:ServiceLevelObjective>
              <wsag:BusinessValueList>...</wsag:BusinessValueList>
        </wsag:GuaranteeTerm>

        <wsag:GuaranteeTerm wsag:name="garantieOfferte" wsag:Obligated="ServiceProvider">
              <wsag:ServiceScope>...</wsag:ServiceScope>
              <wsag:ServiceLevelObjective>
                <wsag:KPITarget>
                      <wsag:KPIName>Resolution</wsag:KPIName>
                      <wsag:Target>
                        <ws-qml:constraint ws-qml:operator="=">
                              <ws-qml:value>
                                10
                                <ws-qml:unit ws-qml:name="megapixels"
                                      ws-qml:semantic-concept="http://www.w3.org/2007/ont/unit#megapixels"/>
                                <ws-qml:type ws-qml:name="decimal"
                                      ws-qml:semantic-concept="http://www.w3c.org/2001/XMLSchema:integer"/>
                              </ws-qml:value>
                        </ws-qml:constraint>
                      </wsag:Target>
                </wsag:KPITarget>
              </wsag:ServiceLevelObjective>
              <wsag:BusinessValueList>...</wsag:BusinessValueList>
        </wsag:GuaranteeTerm>

        <wsag:GuaranteeTerm wsag:name="garantieRequise" wsag:Obligated="ServiceConsumer">
              <wsag:ServiceScope>...</wsag:ServiceScope>
              <wsag:ServiceLevelObjective>
                <wsag:KPITarget>
                      <wsag:KPIName>Bandwidth</wsag:KPIName>
                      <wsag:Target>
                        <ws-qml:constraint ws-qml:operator=">">
                              <ws-qml:value>
                                100
                                <ws-qml:unit ws-qml:name="mbs"
                                      ws-qml:semantic-concept="http://www.w3.org/2007/ont/unit#megabitsps"/>
                                <ws-qml:type ws-qml:name="decimal"
                                      ws-qml:semantic-concept="http://www.w3c.org/2001/XMLSchema:decimal"/>
                              </ws-qml:value>
                        </ws-qml:constraint>
                      </wsag:Target>
                </wsag:KPITarget>
              </wsag:ServiceLevelObjective>
              <wsag:BusinessValueList>...</wsag:BusinessValueList>
        </wsag:GuaranteeTerm>

        <wsag:GuaranteeTerm wsag:name="garantieOfferte" wsag:Obligated="ServiceProvider">
              <wsag:ServiceScope>...</wsag:ServiceScope>
              <wsag:ServiceLevelObjective>
                <wsag:KPITarget>
                      <wsag:KPIName>BandePassante</wsag:KPIName>
                      <wsag:Target>
                        <ws-qml:constraint ws-qml:operator=">">
                              <ws-qml:value>
                                20
                                <ws-qml:unit ws-qml:name="MoS"
                                      ws-qml:semantic-concept="http://www.w3.org/2007/ont/unit#megabytesps"/>
                                <ws-qml:type ws-qml:name="decimal"
                                      ws-qml:semantic-concept="http://www.w3c.org/2001/XMLSchema:decimal"/>
                              </ws-qml:value>
                        </ws-qml:constraint>
                      </wsag:Target>
                </wsag:KPITarget>
              </wsag:ServiceLevelObjective>
              <wsag:BusinessValueList>...</wsag:BusinessValueList>
        </wsag:GuaranteeTerm>
        ...
      </wsag:all>
  </wsag:Terms>
</wsag:Agreement>
```

Code 9.5 – Extrait de WS-Agreement négocié : contraintes.

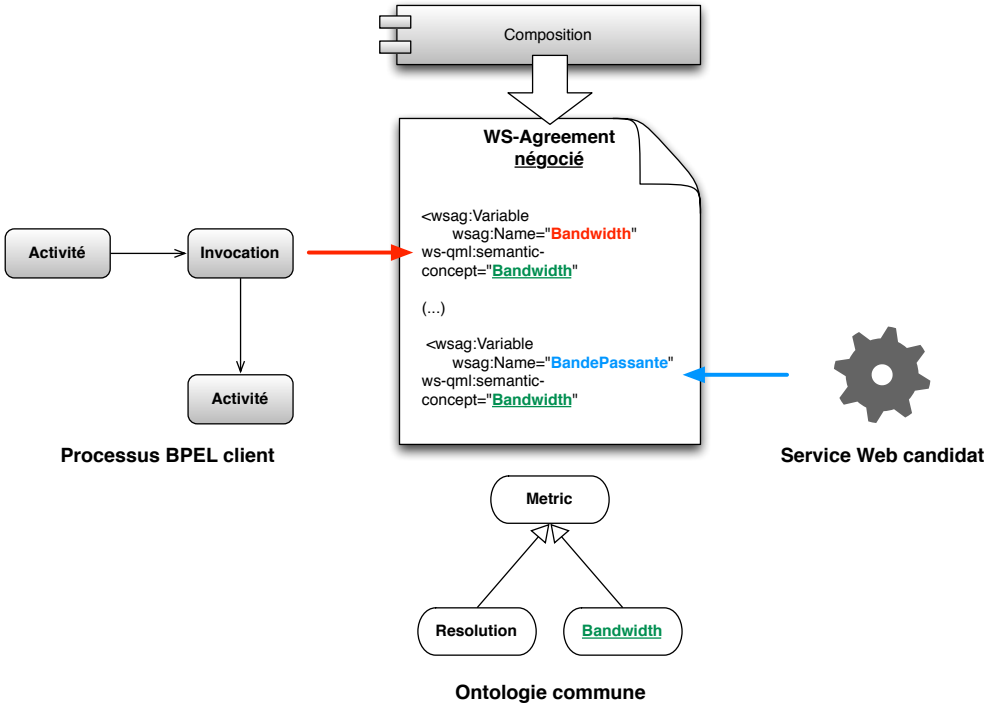

FIGURE 9.15 – Correspondance ontologique entre requis et offert dans un WS-Agreement.

Intégrations des valeurs de supervision lors de la sélection de services

En ce qui concerne les modalités d'échange avec le canevas M4ABP, il a déjà été établi au cours de ce manuscrit (cf. section 5.5.2), que ce dernier doit répondre d'un certain nombre de contraintes. Certaines de ces contraintes se révèlent naturellement dans l'implantation actuelle : celle de *flexibilité dans l'accès aux données* a été intégrée dans le canevas par la gestion de l'hétérogénéité des vocabulaires présentée ci-dessus.

Par ailleurs, l'activité *lateBindingInvoke* étant déclenchée au cours de l'orchestration de service, la *performance dans l'accès au données* de QoS est primordiale pour permettre alors une sélection de service, effectuée sur la base de la QoS courante et des préférences utilisateur, dont l'impact sur le processus global d'orchestration est limité. M4ABP intègre cette contrainte en transmettant les données de QOS depuis son initialisation, en mode *"push"*, vers les abstractions de *vues* coté client, où elles sont "bufferisées". Lors d'un accès à la vue pour obtenir une valeur de QoS précise, la valeur courante stockée sera alors fournie sans le délai lié à la mesure et l'accès réseau.

Pour assurer un certain niveau de *cohérence entre les valeurs de QoS* recueillies par l'activité *lateBindingInvoke via* les *vues*, les données collectées sur les services par le canevas de supervision sont horodatées avec une horloge globale (d'une précision suffisante) et sont filtrées en fonction des paramètres de QoI fixés préalablement par l'activité *lateBindingConfigure* : seules les valeurs comprises dans une certaine fenêtre temporelle sont alors transmises par M4ABP au canevas de liaison tardive. Si l'implantation actuelle limite la QoI à cette notion de fenêtre temporelle, la possibilité de traiter conjointement la "fraîcheur" des données sera assurée à terme.

9.4 Conclusion

Dans ce chapitre, nous avons abordé la mise en œuvre du canevas de modélisation et décision sur LCP-nets, ainsi que celle de la liaison tardive de services Web, au travers des différents choix techniques qui ont dû être effectués pour respecter les contraintes propres au contexte SSOA et à SemEUsE. Cependant, un effort particulier a été effectué pour permettre de futures évolutions : si les choix effectués ici correspondent effectivement à un besoin précis et ponctuel, l'implantation de solutions trop *ad hoc* a pu être évitée dans certains cas, et plus particulièrement en ce qui concerne le canevas des LCP-nets. En effet, ces derniers peuvent être par nature, et comme nous l'avons déjà vu, appliqués à d'autre cadres de décision multi-critères que celui de la sélection de services Web en fonction de leurs valeurs courantes de QoS.

Chapitre 10

Conclusion et perspectives

A l'issue de ces trois années de thèse, il est maintenant possible de dresser un bilan du travail accompli. Effectuée sous convention CIFRE, cette thèse a eu pour point de départ les problématiques propres aux grands systèmes répartis d'aujourd'hui, tels que ceux manipulés couramment dans le service SC2 de mon entreprise d'accueil, Thales Communication France, et dont l'équipe MoVe du LIP6 possède une expertise reconnue. Le projet industriel et académique SemEUsE a ensuite permis de lui donner un cadre, le contexte SSOA, et un cas d'utilisation bien précis, la gestion de crise environnementale.

Dans la suite de ce chapitre, on présente tout d'abord un récapitulatif des contributions avancées par cette thèse et du travail réalisé dans son ensemble, suivi des perspectives à court et à long terme que nous avons identifiées.

Travail réalisé

Les travaux présentés dans cette thèse ont été réalisés pour répondre à une problématique bien précise : comment mettre en œuvre une composition de services particulièrement dynamique, souple, et à même de prendre en compte de multiples contraintes non-fonctionnelles. Nous articulons alors notre démarche autour des concepts de composition *active*, *utile* et *agile*.

Active, de manière à démarquer cette première contribution des approches déjà existantes en terme de composition *dynamique* de services. En effet, nous proposons un mécanisme à même de compléter efficacement ces approches plus traditionnelles pour améliorer les capacités d'adaptation aux changements des systèmes répartis complexes (cf. chapitre 5). Ce mécanisme repose sur une transposition du principe de liaison tardive, bien connu en programmation par objet, au contexte des SSOA. Il s'insère dans un canevas plus général, capable d'effectuer le filtrage des offres de services disponibles dans des annuaires ainsi que de mettre en place la supervision des services de manière à disposer de mesures de QoS.

Ainsi, lors de l'exécution d'un processus métier utilisant les abstractions de programmation que nous avons mises au point, notre canevas de composition active est en mesure de lier tardivement et efficacement un site d'appel, consommateur de service, à un producteur effectif de service. Le

fait de procéder tardivement aux décisions de liaison permet notamment de s'assurer que le service retenu pour répondre au besoin du processus est disponible et que ce sont ses valeurs *courantes* de QoS qui ont été prises en compte lors de la décision. Il s'agit là d'une différence fondamentale par rapport aux approches de composition dynamique (cf. chapitre 3) qui procèdent généralement à une première passe de composition *statique* des services avant l'exécution du processus métier, sur la base de *contraintes* non-fonctionnelles : une fois cette composition fixée, les nouveaux services disponibles ainsi que les variations de QoS des services déjà composés ne seront pris en considération que lors d'une erreur "bloquante" pour la bonne exécution du processus. Il s'agit en fait d'un mécanisme assimilable à une gestion d'exceptions.

Par ailleurs, dans le cadre de cette thèse, la liaison tardive de service sous contraintes non-fonctionnelles a été appliquée au cas d'utilisation de la gestion de crise environnementale, et plus précisement à la lutte anti-incendie dans un contexte civil, sur des feux de grande envergure en milieux naturels (cf. chapitre 4). Il constitue en effet, de par ses caractéristiques intrinsèques, un excellent support à la présentation de cette contribution.

Utile, car la maximisation de l'*utilité* de chaque liaison de service est d'importance et va conditionner la performance globale d'un processus métier lors de son exécution. La sémantique que nous accordons à cette notion d'utilité est celle qui a été mise en avant dans notre nouveau formalisme LCP-net pour l'expression de préférences utilisateur (cf. chapitre 6). Ces préférences vont alors rentrer en compte lors de décisions de liaison.

En effet, afin de permettre la sélection des services dans le contexte multi-critères dans lequel nous nous trouvons, ce formalisme est appliqué à l'élicitation de préférences non-fonctionnelles, établies graphiquement *entre* les propriétés de QoS des services et leurs valeurs (l'utilisateur manipule alors de simples nœuds, arcs et tables). Une fois élicitées, elles vont permettre d'obtenir un ordre total sur chaque ensemble de services candidats, fondé sur l'utilité proportionnelle de la liaison de chacun de ces services à son consommateur respectif.

Bien qu'indépendants de tout domaine spécifique d'application, les LCP-nets s'avèrent particulièrement adaptés au contexte des SOA Sémantiques puisqu'ils permettent aux utilisateurs de partitionner efficacement les domaines continus des variables de Qualité de Service avec des termes linguistiques appropriés, ainsi que d'exprimer l'utilité des affectations d'une manière qualitative plutôt que numérique, cette dernière s'avérant souvent artificielle. Toutefois, le langage LCP-net et le canevas d'inférence que nous avons implanté sur ses bases (cf. chapitre 9) pourraient tout à fait être déployés et appliqués à d'autres contextes de décision multi-critères exhibant des contraintes similaires.

Dans cette optique, nous avons consolidé ce langage *via* la proposition d'une formalisation (cf. chapitre 8). Elle a pour but d'en assurer la pérennité et la réutilisabilité et est effectuée à travers un ensemble de notations et règles de calculs. On montre ainsi que, mathématiquement, un LCP-net est un tuple constitué de variables linguistiques, de trois types d'arcs, de deux types de tables et d'un vecteur de poids.

Agile, car nos contributions sur la composition active et utile de services prennent tout leur sens lorsque déployées conjointement au sein d'une même approche. La contribution agile se présente donc comme la "somme" des deux précédentes (cf. chapitre 7) : s'il s'agit essentiellement de l'utilisation

habile des préférences utilisateurs LCP-net précédemment introduites au moment de la liaison tardive des services, des problématiques bien spécifiques ont du être étudiées et résolues.

Ainsi, c'est dans le cadre de cette mise en œuvre commune qu'est apparu le besoin de définition incrémentale des préférences. Cette constatation a conduit à la définition de la notion de *fragment* de préférence et à l'introduction d'une représentation XML de plus haut niveau des LCP-nets : les *HL*-LCP-nets et *HL*-LCP-frags. Leur formalisation a permis de traiter finement les problèmes d'intégration de ces préférences dans un langage hôte, en l'occurrence BPEL.

Dans le cadre de la composition agile, il nous est aussi apparu utile de proposer une intégration plus poussée des préférences dans les processus métiers BPEL ; l'idée étant de tendre vers des LCP-nets considérés comme des entités de plein droit au cœur de ces processus. Grâce à la généricité de ce formalisme, il nous a été aussi possible d'intégrer des propriétés de QoS globale des processus dans les préférences, et donc, par la coordination agile, dans la décision de liaison elle-même. Cette capacité complète de manière pertinente les travaux sur la composition dynamique en permettant de tenir compte à l'exécution de la réalité du déroulement du processus dans les choix à faire subséquemment : par exemple en permettant de choisir un service plus lent mais plus précis si par ailleurs les appels précédents se sont déroulés plus rapidement que prévu. La composition dynamique plus classique ne permet pas de réaliser ce genre de compromis, or il apparaît que les contraintes de QoS des consommateurs de services sont le plus souvent pessimistes en comparaison de la réalité de l'exécution.

Perspectives à court terme

A court terme, différentes perspectives de travail, pour lesquelles des réflexions préliminaires ont été menées, s'offrent à nous : qu'il s'agisse d'étendre la validation expérimentale de nos contributions, de généraliser le concept de liaison tardive à d'autres cadres techniques, ou bien de poser les bases d'une construction valide de LCP-nets.

Sur une validation expérimentale étendue

Dans le cadre de cette thèse, le principe de composition active de services que nous avons introduit a été appliqué au cas d'utilisation très spécifique de la gestion d'incendies de grande envergure. Afin d'en effectuer une validation extensive, un travail rigoureux d'étude du comportement du composant de liaison tardive de services Web lorsque confronté à un éventail plus large de caractéristiques réseau doit être entrepris. Cette étude est d'autant plus nécessaire qu'il s'agit le plus souvent des conditions usuelles, liées à la nature du réseau en général et du Web en particulier, auxquelles sont soumis les systèmes répartis. Ces conditions peuvent par exemple induire des délais importants de livraison des valeurs de QoS, issues du canevas de supervision, et indispensables à la prise de décision de liaison.

Mener à bien cette validation qui, par nature ne saurait être exhaustive, nécessiterait de développer de nouveaux cas d'utilisation et applications de notre réalisation logicielle. Approche qui devrait être facilitée par la nature même du canevas : bien qu'il ait été mis au point dans un contexte précis, ses principes clés de liaison tardive et décision multi-critères sur préférences utilisateur ne sont pas liés à un cas d'utilisation spécifique.

Au cours de ces expérimentations, le recueil de mesures de performances et l'observation des comportements effectifs des composants de liaison et décision devrait permettre d'étayer la validation de nos contributions.

Sur la généralisation du concept de liaison tardive à d'autres cadres techniques

Cette perspective est d'une nature comparable à celle que nous venons d'évoquer : il s'agit cependant ici, non plus de généraliser nos réalisations à d'autres cas d'utilisation, mais bien à d'autres cadres techniques et technologiques. En effet, nous nous sommes placés, tout au long de cette thèse, en grande partie pour des raisons industrielles, dans le cadre des SSOA. Ce travail, bien que long et fastidieux, serait alors très utile.

Il serait alors raisonnable de généraliser la notion de service et d'appliquer la liaison tardive à d'autres catégories d'applications ou systèmes répartis, des plus basiques et centralisés (de type client/serveur), aux plus évolués et distribués (par exemple de type pair-à-pair) ; tout en explorant d'autres technologies, des objets répartis RMI aux composants répartis CORBA.

Cependant, si nous avons des raisons de croire que ce concept de liaison tardive pourrait être effectivement transposé à d'autres contextes techniques, l'intérêt de cette transposition n'est que plus grand si elle est suivie de cas concrets d'utilisation et de déploiement.

Sur la construction de LCP-nets valides

Les opérations proposées et implémentées par la VM (cf. section 9.2.2) ne garantissent pas l'obtention de LCP-nets *valides* car cette dernière s'attache à construire le plus efficacement possible la représentation en mémoire des LCP-nets. Par exemple, l'ajout d'un fragment de LCP-net à un LCP-net existant, volontairement, ne manipule pas uniquement des LCP-nets valides. Vérifier a posteriori la validité des LCP-nets n'est pas chose facile. Une approche alternative consisterait à définir un nouvel ensemble d'opérateurs élémentaires qui, ne manipulant et retournant que des LCP-nets valides, garantiraient la construction de LCP-nets finaux toujours valides. Ainsi, un LCP-net *atomique* valide serait constitué seulement d'un nœud avec sa CPT associée. Puis, nous définirions les opérateurs élémentaires pour lesquels on préciserait des préconditions, postconditions et des invariants. Par conséquent, considérons :

- n un nœud et V sa variable linguistique ;
- SN l'ensemble des nœuds ;
- (s, t) un arc avec s le nœud origine et t le nœud destination. Dans le cas des ci-arcs, s peut être échangé avec t ;
- SA l'ensemble des arcs (cp, i et ci) : $SA = \mathsf{cp} \cup \mathsf{i} \cup \mathsf{ci}$;
- CPT l'ensemble des CPTs et CIT l'ensemble des CITs. Pour un $cpt \in CPT$, $\dim(cpt) = p$ signifie que cpt a p dimensions ;
- SL l'ensemble des LCP-nets.

Les invariants de ces opérateurs seraient :

- Le nombre total d'arcs (cp, i, ci) ne dépasse pas le nombre de couples (s, t) où $(s, t) \in SN$ et $s \neq t$;
- Exclusion mutuelle des types d'arcs :

  - si $(s,t) \in \mathsf{cp}$ alors $(s,t) \notin \mathsf{i}$ et $(s,t) \notin \mathsf{ci}$;
  - si $(s,t) \in \mathsf{i}$ alors $(s,t) \notin \mathsf{cp}$ et $(s,t) \notin \mathsf{ci}$;
  - si $(s,t) \in \mathsf{ci}$ alors $(s,t) \notin \mathsf{i}$ et $(s,t) \notin \mathsf{cp}$.
- Le nombre de dimensions de la CPT associée au nœud s est égal à 1 + le nombre de cp-arcs entrant dans s ;
- La taille de chaque dimension d'une CPT est inférieure ou égale au nombre de valeurs du domaine de la variable linguistique associée ;
- Il n'y a pas de cycles conditionnels dans le graphe ;
- Il y a au moins un nœud (donc au moins une CPT) et de 0 à n arcs ;
- Il y a au moins autant de tables que de nœuds, c'est-à-dire exactement nb_noeuds CPT et nb_ciarcs CIT.

Par exemple, mettre en œuvre l'opérateur élémentaire d'addition d'un nœud reviendrait à ajouter un nouvel élément à prendre en compte dans le graphe de préférences, c'est-à-dire créer un nœud, la CPT associée, ainsi qu'un ou plusieurs arcs (plus précisément de 1 à k, avec k le nombre de nœuds liés).

Ses préconditions seraient les suivantes :
- un nouveau nœud n à ajouter : $n \notin SN$;
- il existe au moins un LCP-net : $\exists \mathcal{L} \in SL$.

Ses postconditions seraient les suivantes :
- le nœud est ajouté : $n \in SN$;
- une nouvelle CPT est attachée au nœud : $\exists cpt \in CPT$ avec $\dim(cpt) = 1$.
- (au moins) un nouvel arc apparaît : $\exists (s,t) \in SA$;
- le (les) LCP-net(s) est (sont) modifié(s) : $\mathcal{L} \notin SL, \exists \mathcal{L}' \in SL$.

Ce travail pourrait nous amener, à plus long terme, à définir une sorte d'algèbre des LCP-nets.

Perspectives à long terme

On présente ici des perspectives plus larges, qui s'inscrivent dans la prolongation de nos contributions, mais pour lesquelles une étude plus approfondie doit être menée. Elles ouvrent, nous l'espérons, de nouveaux angles de recherche.

Sur un approfondissement du travail autour des LCP-nets

L'algèbre des LCP-nets évoquée ci-dessus permettrait de reprendre cette formalisation dans un cadre théorique plus large et complètement indépendant du cas d'utilisation de cette thèse. Typiquement, un tel travail nous amènerait à définir un groupe abélien $(SL, *)$ avec $*$ une loi de composition interne. Concrètement, la composition de deux (ou plus) LCP-nets serait pertinente dans le cas où l'on souhaite combiner deux (ou plus) réseaux de préférences concernant la même décision. Par exemple, deux utilisateurs doivent utiliser le même service à choisir parmi un ensemble, avec des préférences communes : il serait nécessaire, dans ce cas, de combiner les deux LCP-nets définis par les deux clients afin d'arriver à une décision conjointe.

Par ailleurs, une amélioration intéressante serait de comparer la rapidité des calculs si l'on considère uniquement des 2-tuples au sein des LCP-nets, c'est-à-dire si l'on propose un traitement lin-

guistique de bout en bout. En effet, lors de l'utilisation des 2-tuples (que ce soit le formalisme d'Herrera et Martínez, de Wang et Hao ou de Truck et Akdag), on s'affranchit, en tous cas, dans les calculs, des sous-ensembles flous, tout en gardant une sémantique linguistique attachée aux objets manipulés. Il faudrait donc voir si ces calculs numériques sont moins lourds que ceux associés aux SEFs. Cela semble probable vu que seuls des calculs sur des indices, ou bien des implications de type $\min(1 - x + y, 1)$ sont réalisés, alors que le traitement avec manipulation de SEFs de bout en bout implique des calculs d'intersection de droites (si les SEFs sont trapézoïdaux), qui peuvent être considérés, dans l'absolu, comme plus longs.

Cela impliquerait un travail conséquent sur la modélisation d'une inférence *ad hoc* à ces formalismes 2-tuples, piste par ailleurs intéressante, indépendamment de nos LCP-nets.

Sur une coordination des prises de décision de liaison par le partage d'agents de décision

Cette thèse a introduit des abstractions de programmation permettant d'effectuer, lors de la coordination agile de service, de multiples décisions *locales* de liaison entre consommateurs et producteurs de services : cette prise de décision est actuellement effectuée *ceteris paribus* à chaque site d'appel de service d'un processus métier, en fonction de ses préférences utilisateur locales et des valeurs courantes de QoS des services considérés ; c'est-à-dire indépendamment des autres décisions passées ou à venir.

La transition d'un ensemble de décisions locales *indépendantes*, à une coordination de cet ensemble au niveau du processus, se justifie par la problématique d'une gestion efficace de sa *QoS globale*. Une première étape vers la prise en compte de ces contraintes a été proposée dans cette thèse par l'utilisation des variables des LCP-nets comme moyen de réintroduction des valeurs courantes de QoS globale des processus dans les décisions de liaison (cf. section 7.5). Cependant, malgré cette réflexivité, ces décisions restent essentiellement locales.

Il apparaît alors clairement qu'une *décision globale et coordonnée* (sur le processus dans son ensemble) va nécessiter l'introduction d'abstractions de programmation spécifiques, car celle-ci ne doit pas se faire au détriment de l'agilité de la composition, comme c'est le cas dans les approches classiques (cf. section 3.3). Se pose alors la question du choix d'une abstraction adaptée, à laquelle le principe de factorisation de LCP-net d'un processus que nous avons déjà abordé (cf. section 7.2.2) apporte un élément de réponse. Il semble en effet judicieux d'accompagner cette factorisation des modèles de décision par une factorisation des *agents de décision* eux-mêmes. Si l'implantation actuelle fait usage d'agents de décision locaux à chaque site d'appel de service, ils pourraient en être extraits et *partagés entre les sites*.

La possibilité d'utiliser conjointement ces agents de décision globaux, indépendants de la structure du programme, et une approche de décision séquentielle [Littman, 1996] pourrait alors constituer un axe de développement de nos travaux sur le long terme.

Index

Bibliographie

[Aagedal, 2001] Aagedal, J. (2001). *Quality of service support in development of distributed systems*. Thèse d'université, Department of Informatics, Faculty of Mathematics and Natural Sciences, University of Oslo.

[Alcalá et al., 2007] Alcalá, R., Alcalá-Fdez, J., Herrera, F., et Otero, J. (2007). Genetic learning of accurate and compact fuzzy rule based systems based on the 2-tuples linguistic representation. *International Journal of Approximate Reasoning*, 44(1) : 45–64.

[Andrieux et al., 2004] Andrieux, A., Czajkowski, K., Dan, A., Keahey, K., Ludwig, H., Pruyne, J., Rofrano, J., Tuecke, S., et Xu, M. (2004). The design of gccl : a generalized common contract language. Rapport technique, Global Grid Forum.

[Bacchus et Grove, 1995] Bacchus, F. et Grove, A. (1995). Graphical models for preference and utility. In *Proceedings of the Eleventh Conference on Uncertainty in Artificial Intelligence*, pages 3–10.

[Bajaj et al., 2006] Bajaj, S., Box, D., Chappell, D., Curbera, F., Daniels, G., Hallam-Baker, P., Hondo, M., Kaler, C., Langworthy, D., Malhotra, A., et al. (2006). Web services policy framework (ws-policy). Rapport technique, IBM, BEA Systems, Microsoft, SAP AG, Sonic Software, VeriSign.

[Baligand et al., 2007] Baligand, F., Rivierre, N., et Ledoux, T. (2007). A declarative approach for qos-aware web service compositions. In *Proceedings of ICSOC*, pages 422–428.

[Ben Mokhtar et al., 2007] Ben Mokhtar, S., Georgantas, N., et Issarny, V. (2007). COCOA : COnversation-based service COmposition in pervAsive computing environments with QoS support. *The Journal of Systems & Software*, 80(12) : 1941–1955.

[Ben Mokhtar et al., 2005] Ben Mokhtar, S., Liu, J., Georgantas, N., et Issarny, V. (2005). Qos-aware dynamic service composition in ambient intelligence environments. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, pages 317–320. ACM.

[Berbner et al., 2007] Berbner, R., Spahn, M., Repp, N., Heckmann, O., et Steinmetz, R. (2007). Wsqosx - a qos architecture for web service workflows. In *Proceedings of ICSOC*, pages 623–624.

[Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J., et Lassila, O. (2001). The semantic web. *Scientific American*, 284(5) : 34–43.

[Beugnard et al., 1999] Beugnard, A., Jézéquel, J.-M., et Plouzeau, N. (1999). Making components contract aware. *IEEE Computer*, 32(7) : 38–45.

[Boloni et al., 1997] Boloni, L., Jun, K., et Marinescu, D. (1997). QoS and Reliability Models for Network Computing. Rapport technique, Department of Computer Sciences, Purdue University.

[Boubekeur et Tamine-Lechani, 2006] Boubekeur, F. et Tamine-Lechani, L. (2006). Recherche d'information flexible basée CP-Nets. In *Proceedings of CORIA'06*, pages 161–167.

[Bouchon-Meunier, 1995] Bouchon-Meunier, B. (1995). *La logique floue et ses applications*. Addison-Wesley France, Paris.

[Boutilier et al., 2001] Boutilier, C., Bacchus, F., et Brafman, R. I. (2001). UCP-Networks : A directed graphical representation of conditional utilities. In *Proceedings of the Seventeenth Conference on Uncertainty in Artificial Intelligence*, pages 56–64.

[Boutilier et al., 2004] Boutilier, C., Brafman, R. I., Domshlak, C., Hoos, H. H., et Poole, D. (2004). CP-nets : A tool for representing and reasoning with conditional *Ceteris Paribus* Preference Statements. *Journal of Artificial Intelligence Research*, 21 : 135–191.

[Bouveret et al., 2005] Bouveret, S., Lemaître, M., Fargier, H., et Lang, J. (2005). Allocation of indivisible goods : a general model and some complexity results. In *Proceedings of AAMAS*, pages 1309–1310.

[Brafman et Domshlak, 2002] Brafman, R. I. et Domshlak, C. (2002). Introducing variable importance tradeoffs into CP-nets. In *Proceedings of the Eighteenth Annual Conference on Uncertainty in Artificial Intelligence*, pages 69–76.

[Braziunas et Boutilier, 2005] Braziunas, D. et Boutilier, C. (2005). Local utility elicitation in GAI models. In *Proceedings of UAI*, pages 42–49.

[Buckley, 1984] Buckley, J. (1984). Multiple judge, multiple criteria ranking problem : A fuzzy set approach. *Fuzzy Sets and Systems*, 13(1) : 25–38.

[Budinsky et al., 2003] Budinsky, F., Brodsky, S., et Merks, E. (2003). *Eclipse modeling framework*. Pearson Education.

[Canfora et al., 2006] Canfora, G., Penta, M. D., Esposito, R., Perfetto, F., et Villani, M. L. (2006). Service composition (re)binding driven by application-specific qos. In *Proceedings of ICSOC*, pages 141–152.

[Canfora et al., 2005a] Canfora, G., Penta, M. D., Esposito, R., et Villani, M. L. (2005a). An approach for qos-aware service composition based on genetic algorithms. In *Proceedings of GECCO*, pages 1069–1075.

[Canfora et al., 2005b] Canfora, G., Penta, M. D., Esposito, R., et Villani, M. L. (2005b). Qos-aware replanning of composite web services. In *Proceedings of ICWS*, pages 121–129.

[Cardoso, 2002] Cardoso, J. (2002). Quality of service and semantic composition of workflows. Rapport technique, Department of Computer Science. Athens, GA, University of Georgia.

[Cardoso et al., 2004] Cardoso, J., Sheth, A. P., Miller, J. A., Arnold, J., et Kochut, K. (2004). Quality of service for workflows and web service processes. *J. Web Sem.*, 1(3) : 281–308.

[Castro, 1995] Castro, J. (1995). Fuzzy logic controllers are universal approximators. *Ieee Transactions on Systems Man and Cybernetics*, 25(4) : 629–635.

[Chafle et al., 2006] Chafle, G., Dasgupta, K., Kumar, A., Mittal, S., et Srivastava, B. (2006). Adaptation in web service composition and execution. In *Proceedings of ICWS, Industry Track*.

[Châtel, 2007a] Châtel, P. (2007a). Toward a Semantic Web service discovery and dynamic orchestration based on the formal specification of functional domain knowledge. In *Proceedings of ICSSEA'07*.

[Châtel, 2007b] Châtel, P. (2007b). Une architecture pour la découverte et l'orchestration de services Web sémantiques. In *Proceedings of the First* Journées Francophones sur les Ontologies *(JFO'07)*, pages 247–264.

[Châtel et al., 2010a] Châtel, P., Malenfant, J., et Truck, I. (2010a). QoS-based Late-Binding of Service Invocations in Adaptive Business Processes. In *Proceedings of ICWS*.

[Châtel et al., 2008] Châtel, P., Truck, I., et Malenfant, J. (2008). A linguistic approach for non-functional preferences in a semantic SOA environment. In *Computational Intelligence in Decision and Control, Proceedings of the 8th International FLINS Conference*, pages 889–894.

[Châtel et al., 2010b] Châtel, P., Truck, I., et Malenfant, J. (2010b). Lcp-nets : A linguistic approach for non-functional preferences in a semantic soa environment. *J.UCS Journal of Universal Computer Science*, 16(1) : 198–217.

[Chevaleyre et al., 2004] Chevaleyre, Y., Endriss, U., Estivie, S., Maudet, N., et al. (2004). Multiagent resource allocation with k-additive utility functions. In *Proceedings of DIMACS-LAMSADE Workshop on Computer Science and Decision Theory*, volume 3, pages 83–100.

[Chinnici et al., 2007] Chinnici, R., Moreau, J., Ryman, A., et Weerawarana, S. (2007). Web services description language (WSDL) version 2.0 part 1 : Core language. Rapport technique, World Wide Web Consortium.

[Chiu et al., 2009] Chiu, D., Deshpande, S., Agrawal, G., et Li, R. (2009). A dynamic approach toward qos-aware service workflow composition. In *Proceedings of ICWS*, pages 655–662. IEEE Computer Society.

[Clark et al., 1999] Clark, J., DeRose, S., et al. (1999). XML path language (XPath) version 1.0. Rapport technique, W3C.

[Cleaveland et Smolka, 1996] Cleaveland, R. et Smolka, S. (1996). Strategic directions in concurrency research. *ACM Computing Surveys (CSUR)*, 28(4) : 607–625.

[Colombo et al., 2006] Colombo, M., Di Nitto, E., et Mauri, M. (2006). Scene : A service composition execution environment supporting dynamic changes disciplined through rules. *Lecture Notes in Computer Science*, 4294 : 191–202.

[Cowell et al., 1999] Cowell, R., Dawid, A., Lauritzen, S., et Spiegelhalter, D. (1999). *Probabilistic networks and expert systems*. Springer Verlag.

[Cumming, 2003] Cumming, M. (2003). Tomatoes are not the only fruit : a rough guide to taxonomies, thesauri, ontologies and the like. Rapport technique, Cabinet Office - UK Government.

[Curbera et Weerawarana, 2001] Curbera, F. et Weerawarana, S. (2001). Web Services Description Language (WSDL) 1.1. Rapport technique, W3C.

[Debreu, 1964] Debreu, G. (1964). Continuity properties of Paretian utility. *International Economic Review*, 5(3) : 285–293.

[Degani et Bortolan, 1988] Degani, R. et Bortolan, G. (1988). The Problem of Linguistic Approximation in Clinical Decision Making. *International Journal of Approximate Reasoning*, 2 : 143–162.

[Delgado et al., 1993] Delgado, M., Verdegay, J., et Vila, M. (1993). On Aggregation Operations of Linguistic Labels. *International Journal of Intelligent Systems*, 8 : 351–370.

[Dobson et al., 2005] Dobson, G., Lock, R., et Sommerville, I. (2005). QoSOnt : a QoS ontology for service-centric systems. In *Proceedings of the 31st EUROMICRO Conference on Software Engineering and Advanced Applications*, pages 80–87. IEEE Computer Society.

[Driankov et al., 1993] Driankov, D., Hellendoorn, H., et Reinfrank, M. (1993). *An introduction to fuzzy control*. Springer-Verlag.

[Eberhart, 2004] Eberhart, A. (2004). Ad-hoc invocation of semantic web services. In *Proceedings of ICWS*, pages 116–123.

[Floyd, 1967] Floyd, R. (1967). Assigning meanings to programs. *Mathematical aspects of computer science*, 19(19–32).

[Frølund et Koistinen, 1998a] Frølund, S. et Koistinen, J. (1998a). Qml : A language for quality of service specification. Rapport technique, Hewlett-Packard Laboratories.

[Frølund et Koistinen, 1998b] Frølund, S. et Koistinen, J. (1998b). Quality-of-service specification in distributed object systems. *Distributed Systems Engineering*, 5 : 179–202.

[Fujii et Suda, 2009] Fujii, K. et Suda, T. (2009). Semantics-based context-aware dynamic service composition. *TAAS*, 4(2).

[Gelenbe et al., 2004] Gelenbe, E., Lent, R., et Nunez, A. (2004). Self-aware networks and QoS. *Proceedings of the IEEE*, 92(9) : 1478–1489.

[Gonzales et Perny, 2004] Gonzales, C. et Perny, P. (2004). Gai networks for utility elicitation. In *Principles of Knowledge Representation and Reasoning : Proceedings of the Ninth International Conference (KR2004)*, pages 224–234.

[Gonzales et Perny, 2005] Gonzales, C. et Perny, P. (2005). GAI networks for decision making under certainty. In *Proceedings of IJCAI-05 Workshop on Advances in Preference Handling*.

[Grabisch, 1997] Grabisch, M. (1997). K-order additive discrete fuzzy measures and their representation. *Fuzzy sets and systems*, 92(2) : 167–189.

[Halima et al., 2008] Halima, R., Drira, K., et Jmaiel, M. (2008). A QoS-Oriented Reconfigurable Middleware for Self-Healing Web Services. In *Proceedings of the 2008 IEEE International Conference on Web Services*, pages 104–111. IEEE Computer Society.

[Halpern, 1993] Halpern, M. (1993). Binding. In Ralston, A. et Reilly, E., éditeurs, *Encyclopedia of Computer Science*, page 125. Chapman & Hall, third edition.

[Herrera et al., 2009] Herrera, F., Alonso, S., Chiclana, F., et Herrera-Viedma, E. (2009). Computing with words in decision making : foundations, trends and prospects. *Fuzzy Optimization and Decision Making*, 8(4) : 337–364.

[Herrera et Herrera-Viedma, 2000] Herrera, F. et Herrera-Viedma, E. (2000). Linguistic decision analysis : steps for solving decision problems under linguistic information. *Fuzzy Sets and Systems*, 115(1) : 67–82.

[Herrera et al., 2001] Herrera, F., Herrera-Viedma, E., et Martínez, L. (2001). A Hierarchical Ordinal Model for Managing Unbalanced Linguistic Term Sets Based on the Linguistic 2-Tuple Model. In *Proceedings of EUROFUSE Workshop on Preference Modelling and Applications*, pages 201–206.

[Herrera et al., 2008] Herrera, F., Herrera-Viedma, E., et Martínez, L. (2008). A fuzzy linguistic methodology to deal with unbalanced linguistic term sets. *IEEE Transactions on Fuzzy Systems*, 16(2) : 354–370.

[Herrera et Martínez, 2000] Herrera, F. et Martínez, L. (2000). A 2-tuple fuzzy linguistic representation model for computing with words. *IEEE Transactions on Fuzzy Systems*, 8(6) : 746–752.

[Herrera et Martínez, 2001] Herrera, F. et Martínez, L. (2001). A model based on linguistic 2-tuples for dealing with multigranular hierarchical linguistic contexts in multi-expert decision-making. *IEEE Transactions on Systems, Man, and Cybernetics, Part B*, 31(2) : 227–234.

[Herrera et al., 2002] Herrera, F., Martínez, L., Herrera-Viedma, E., et Chiclana, F. (2002). Fusion of Multigranular Linguistic Information based on the 2-tuple Fuzzy Linguistic Representation Model. In *Proceedings of IPMU 2002*, pages 1155–1162.

[Herrera et al., 2005] Herrera, F., Martınez, L., et Sánchez, P. (2005). Managing non-homogeneous information in group decision making. *European Journal of Operational Research*, 166(1) : 115–132.

[Hoare, 1969] Hoare, C. A. R. (1969). An axiomatic basis for computer programming. *Commun. ACM*, 12(10) : 576–580.

[IEC, 2001] IEC (2001). IEC 61131-7 Fuzzy Control Programming.

[ISO/IEC, 1998] ISO/IEC (1998). Information technology - quality of service : Framework. iso/iec 13236 :1998, itu-t x.641.

[Jacobson, 1991] Jacobson, I. (1991). *Object-oriented software engineering.* ACM.

[Jacobson et al., 1999] Jacobson, I., Booch, G., et Rumbaugh, J. (1999). *The unified software development process.* Addison-Wesley.

[Jang, 1993] Jang, J. (1993). ANFIS : Adaptive-network-based fuzzy inference system. *IEEE transactions on systems, man, and cybernetics*, 23(3) : 665–685.

[Jiang et al., 2008] Jiang, Y., Fan, Z., et Ma, J. (2008). A method for group decision making with multi-granularity linguistic assessment information. *Information Sciences*, 178(4) : 1098–1109.

[Johansson et al., 1993] Johansson, H., Johansson, H., et Pendlebury, A. (1993). *Business process reengineering : Breakpoint strategies for market dominance.* Wiley New York.

[Kagal, 2002] Kagal, L. (2002). Rei : A policy language for the me-centric project. Rapport technique, HP Labs.

[Keeney et Raiffa, 1993] Keeney, R. et Raiffa, H. (1993). Decisions with multiple objectives. *Cambridge Books.*

[Keller et Ludwig, 2003] Keller, A. et Ludwig, H. (2003). The WSLA framework : Specifying and monitoring service level agreements for web services. *Journal of Network and Systems Management*, 11(1) : 57–81.

[Kopeckỳ et al., 2007] Kopeckỳ, J., Vitvar, T., Bournez, C., et Farrell, J. (2007). Sawsdl : Semantic annotations for wsdl and xml schema. *IEEE Internet Computing*, pages 60–67.

[Kutz et al., 2003] Kutz, O., Lutz, C., Wolter, F., et Zakharyaschev, M. (2003). Basic description logics. *Journal of Logic and Computation*, pages 43–95.

[Le Duc et al., 2009] Le Duc, B., Châtel, P., Rivierre, N., Malenfant, J., Collet, P., et Truck, I. (2009). Non-functional Data Collection for Adaptive Business Process and Decision Making. In *Proceedings of MW4SOC'09 workshop*, pages 7–12. ACM.

[Lee et al., 2003] Lee, K., Jeon, J., Lee, W., Jeong, S., et Park, S. (2003). Qos for web services : Requirements and possible approaches. *W3C Working Group Note*, 25.

[Li et Horrocks, 2004] Li, L. et Horrocks, I. (2004). A software framework for matchmaking based on semantic web technology. *International Journal of Electronic Commerce*, 8(4) : 39–60.

[Littman, 1996] Littman, M. (1996). Algorithms for sequential decision making. Rapport technique, Brown University, Providence, RI.

[Malenfant et al., 2002] Malenfant, J., Plouzeau, N., et Jézéquel, J. (2002). The design of gccl : a generalized common contract language. Rapport technique, INRIA.

[Mamdani et Assilian, 1975] Mamdani, E. et Assilian, S. (1975). An experiment in linguistic synthesis with a fuzzy logic controller. *International Journal of Man-Machine Studies*, 7(1) : 1–13.

[Martin et al., 2004] Martin, D., Burstein, M., Hobbs, J., Lassila, O., McDermott, D., McIlraith, S., Narayanan, S., Paolucci, M., Parsia, B., Payne, T., et al. (2004). OWL-S : Semantic markup for web services. Rapport technique, W3C.

[Maximilien et Singh, 2004] Maximilien, E. et Singh, M. (2004). A framework and ontology for dynamic web services selection. *IEEE Internet Computing*, pages 84–93.

[McGuinness et al., 2004] McGuinness, D., Van Harmelen, F., et al. (2004). OWL web ontology language overview. Rapport technique, W3C.

[Mernik et al., 2005] Mernik, M., Heering, J., , et Sloane, A. M. (2005). When and how to develop domain-specific languages. *ACM Computing Surveys*, 37(4) : 316–344.

[Moreau, 2009] Moreau, A. (2009). *Mise en oeuvre automatique de processus métier dans le domaine des architectures orientées services.* Thèse d'université, Université Paris 6, EDITE de Paris.

[Moreau et al., 2009] Moreau, A., Malenfant, J., et Dao, M. (2009). Data Flow Repair in Web Service Orchestration at Runtime. In *Proceedings of ICIW*, pages 43–48. IEEE Computer Society Press.

[Papazoglou, 2008] Papazoglou, M. P. (2008). *Web services : principles and technology.* Pearson Prentice Hall.

[Papazoglou et Georgakopoulos, 2003] Papazoglou, M. P. et Georgakopoulos, D. (2003). Service-oriented computing. *Communications of the ACM*, 46.

[Patil et al., 2003] Patil, A., Oundhakar, S., et Sheth, A. (2003). Semantic annotation of Web services. Rapport technique, LSDIS Lab, Department of Computer Science, University of Georgia.

[Pearl et Shafer, 1988] Pearl, J. et Shafer, G. (1988). *Probabilistic reasoning in intelligent systems : networks of plausible inference.* Morgan Kaufmann San Mateo, CA.

[Peer, 2002] Peer, J. (2002). Bringing together Semantic Web and Web services. In *Proceedings of the First International Semantic Web Conference*, Sardinia, Italy.

[Petri et Reisig, 2008] Petri, C. et Reisig, W. (2008). Petri nets. *Scholarpedia*, 3 : 6477.

[Pini et al., 2005] Pini, M., Rossi, F., Venable, K., et Walsh, T. (2005). Aggregating partially ordered preferences : impossibility and possibility results. In *Proceedings of the 10th conference on Theoretical aspects of rationality and knowledge*, pages 193–206.

[Puterman, 1994] Puterman, M. (1994). *Markov Decision Processes : Discrete Stochastic Dynamic Programming.* Wiley, New York.

[Queiroz et al., 2007] Queiroz, S., Gonzales, C., et Perny, P. (2007). Décision collective avec des réseaux GAI. In *Congrès de la Société Française de Recherche Opérationnelle et d'Aide à la Décision*, pages 217–227. Presses Universitaires de Grenoble.

[Recommendation, 1995] Recommendation, E. (1995). 800-Telephone Network and ISDN, Quality of Service, Network Management and Traffic Engineering, Terms and Definitions Related to Quality of Service and Network Performance Including Dependability. Rapport technique, International Telecommunication Union, International Telecommunication Union.

[Richters et Gogolla, 2002] Richters, M. et Gogolla, M. (2002). OCL : Syntax, semantics, and tools. *Lecture Notes in Computer Science*, 2263 : 42–68.

[Roman et al., 2005] Roman, D., Keller, U., Lausen, H., de Bruijn, J., Lara, R., Stollberg, M., Polleres, A., Feier, C., Bussler, C., et Fensel, D. (2005). Web service modeling ontology. *Applied Ontology*, 1(1) : 77–106.

[Rossi et al., 2004] Rossi, F., Venable, K. B., et Walsh, T. (2004). mcp nets : Representing and reasoning with preferences of multiple agents. In *Proceedings of the Nineteenth National Conference on Artificial Intelligence, Sixteenth Conference on Innovative Applications of Artificial Intelligence*, pages 729–734.

[Rottger et Zschaler, 2003] Rottger, S. et Zschaler, S. (2003). CQML+ : Enhancements to CQML. In *Proceedings of the 1st International Workshop on Quality of Service in Component-Based Software Engineering*, pages 43–56. Cépaduès-Éditions.

[Roubens, 1997] Roubens, M. (1997). Fuzzy sets and decision analysis. *Fuzzy Sets and Systems*, 90(2) : 199–206.

[Rumbaugh et al., 2004] Rumbaugh, J., Jacobson, I., et Booch, G. (2004). *Unified Modeling Language Reference Manual, The*. Pearson Higher Education.

[Sabata et al., 1997] Sabata, B., Chatterjee, S., Davis, M., Sydir, J., Lawrence, T., Int, S., et Park, M. (1997). Taxonomy for QoS specifications. In *Proceedings of the Third International Workshop on Object-Oriented Real-Time Dependable Systems*, pages 100–107.

[Santhanam et al., 2008] Santhanam, G. R., Basu, S., et Honavar, V. (2008). Tcp-compose* - a tcp-net based algorithm for efficient composition of web services using qualitative preferences. In *Proceedings of ICSOC*, pages 453–467.

[Schröpfer et al., 2007] Schröpfer, C., Binshtok, M., Shimony, S. E., Dayan, A., Brafman, R., Offermann, P., et Holschke, O. (2007). Introducing preferences over NFPs into service selection in SOA. In *Proceedings of NFPSLA-SOC'07*.

[Shapiro, 2002] Shapiro, R. (2002). A technical comparison of xpdl, bpml and bpel4ws. *Cape Visions*.

[Shendrik et Tamm, 1986] Shendrik, M. et Tamm, G. (1986). An approach to interactive solution of multicriterial optimization problems with linquistic modeling of preferences. *Automatic Control and Computer Sciences*, 19(6) : 1–7.

[Sivashanmugam et al., 2003] Sivashanmugam, K., Verma, K., Sheth, A. P., et Miller, J. A. (2003). Adding semantics to web services standards. In *Proceedings of ICWS*, pages 395–401.

[Szyperski et al., 1999] Szyperski, C. A., Bosch, J., et Weck, W. (1999). Component-oriented programming. In *Proceedings of ECOOP Workshops*, pages 184–192.

[Tian et al., 2003] Tian, M., Gramm, A., Naumowicz, T., Ritter, H., et Schiller, J. (2003). A concept for QoS integration in web services. In *Proceedings of WQW2003*, pages 149–155.

[Tong et Bonissone, 1980] Tong, R. et Bonissone, P. (1980). Linguistic Approach to Decision Making With Fuzzy Sets. *IEEE Transactions on Systems, Man, and Cybernetics*, 10(11) : 716–723.

[Tonti et al., 2003] Tonti, G., Bradshaw, J. M., Jeffers, R., Montanari, R., Suri, N., et Uszok, A. (2003). Semantic web languages for policy representation and reasoning : A comparison of kaos, rei, and ponder. In *Proceedings of International Semantic Web Conference*, pages 419–437.

[Tosic et al., 2002] Tosic, V., Patel, K., et Pagurek, B. (2002). Wsol-web service offerings language. *Lecture notes in computer science*, pages 57–67.

[Truck et Akdag, 2006] Truck, I. et Akdag, H. (2006). Manipulation of Qualitative Degrees to Handle Uncertainty : Formal Methods and Applications. *Knowledge and Information Systems (KAIS)*, 9(4) : 385–411.

[Uszok et al., 2004] Uszok, A., Bradshaw, J., Jeffers, R., Johnson, M., Tate, A., Dalton, J., et Aitken, S. (2004). Policy and contract management for semantic web services. In *Proceedings of AAAI 2004 Spring Symposium Workshop on Knowledge Representation and Ontology for Autonomous Systems*.

[Van der Aalst et Ter Hofstede, 2005] Van der Aalst, W. et Ter Hofstede, A. (2005). YAWL : yet another workflow language. *Information Systems*, 30(4) : 245–275.

[Verma et al., 2005] Verma, K., Akkiraju, R., et Goodwin, R. (2005). Semantic matching of Web service policies. In *Proceedings of the Second Workshop on SDWP*, pages 79–90.

[Vitvar et al., 2007] Vitvar, T., Moran, M., Zaremba, M., Haller, A., et Kotinurmi, P. (2007). Semantic SOA to promote integration of heterogeneous B2B services. In *Proceedings of IEEE CEC'07 and EEE'07*.

[Wang et Hao, 2006] Wang, J. et Hao, J. (2006). A new version of 2-tuple fuzzy linguistic representation model for computing with words. *IEEE Transactions on Fuzzy Systems*, 14(3) : 435–445.

[Welty et McGuinness, 2004] Welty, C. et McGuinness, D. (2004). Owl web ontology language guide. *World Wide Web Consortium (W3C) Recommendation, Feb*.

[White, 2004a] White, S. (2004a). Introduction to BPMN. Rapport technique, IBM Cooperation.

[White, 2004b] White, S. (2004b). Process modeling notations and workflow patterns. In *Proceedings of BPTrends*, pages 265–294.

[Wu et Mendel, 2007] Wu, D. et Mendel, J. (2007). Aggregation using the linguistic weighted average and interval type-2 fuzzy sets. *IEEE Transactions on Fuzzy Systems*, 15(6) : 1145–1161.

[Xu, 2004] Xu, Z. (2004). A method based on linguistic aggregation operators for group decision making with linguistic preference relations. *Information Sciences*, 166(1-4) : 19–30.

[Xu, 2008] Xu, Z. (2008). *Linguistic Aggregation Operators : An Overview*, volume 220, pages 163–181. Springer Verlag. ISBN : 978-3-540-73722-3.

[Yager, 1981] Yager, R. (1981). A new methodology for ordinal multiple aspect decisions based on fuzzy sets. *Decision Sciences*, 12(58).

[Yager, 1988] Yager, R. (1988). On ordered weighted averaging aggregation operators in multicriteria decisionmaking. *IEEE Trans. Syst. Man Cybern.*, 18(1) : 183–190.

[Yager, 2007] Yager, R. (2007). Using Stress Functions to Obtain OWA Operators. *IEEE Trans. on Fuzzy Systems*, 15(6) : 1122–1129.

[Zadeh, 1975] Zadeh, L. (1975). The Concept of a Linguistic Variable and Its Applications to Approximate Reasoning. *Information Sciences, Part I, II, III*, 8,8,9 : 199–249, 301–357, 43–80.

[Zadeh, 1996] Zadeh, L. (1996). Fuzzy logic = computing with words. *IEEE Transactions on Fuzzy Systems*, 4 : 103–111.

[Zeng et al., 2004] Zeng, L., Benatallah, B., Ngu, A., Dumas, M., Kalagnanam, J., et Chang, H. (2004). QoS-aware middleware for web services composition. *IEEE Transactions on software engineering*, 30(5) : 311–327.

[Zhou et al., 2004] Zhou, C., Chia, L.-T., et Lee, B.-S. (2004). Daml-qos ontology for web services. In *Proceedings of ICWS*, pages 472–479.

Résumé

La mise en œuvre d'une composition de services, dans le contexte de l'entreprise et du Web, ouvre la perspective de nombreux champs d'investigations et d'améliorations. Les contributions de cette thèse ont alors pour vocation de réaliser une composition particulièrement dynamique, souple, et capable de prendre en compte de multiples contraintes non-fonctionnelles. Elles s'articulent autour des concepts de composition active, utile et agile.

La **composition active** est à même de compléter efficacement les approches *dynamiques* classiques, pour en améliorer les capacités d'adaptation aux changements non-fonctionnels. Elle repose sur une transposition du principe de liaison tardive au contexte des SSOA, alors disposé pour intégrer la *QoS courante* des services à l'exécution. La **composition utile** est liée à notre nouveau formalisme *LCP-net* pour l'expression de préférences utilisateur. L'élicitation de préférences non-fonctionnelles, établies *entre* les propriétés de QoS des services et leurs valeurs, permet d'obtenir un ordre total ou quasi total sur chaque ensemble de services candidats lors de leur sélection. Enfin, la **composition agile** correspond à la "somme" des deux précédentes : il s'agit d'une utilisation habile des préférences utilisateurs LCP-net lors de la liaison tardive des services. Nous y posons aussi les bases d'une gestion de la QoS globale des processus métiers.

Mots-clés: Architecture Orientée Services, composition de services, contrainte non-fonctionnelle, approche qualitative, approche linguistique, décision multi-critères, logique floue, modélisation de préférences, liaison tardive.

Abstract

Service composition implementation, in a Web and business context, opens many investigation and improvements prospects. The contributions in this thesis are then intended to perform a particularly dynamic and flexible composition, able to take into account multiple non-functional constraints. They revolve around the concepts of active, useful and agile composition.

Active composition is able to effectively complement dynamic approaches, in order to improve their capability to adapt to non-functional changes. It relies on a transposition of the late-binding principle to the SSOA context, then able to integrate the current QoS values of services at execution time. **Useful composition** is linked to our new *LCP-net* formalism for expressing user preferences. The elicitation of non-functional preferences, established *between* services QoS properties and their values, will afterward provide a total or almost total order over each candidate services sets, during service selection. Finally, **agile composition** is the "sum" of the last two, where we lay the groundwork for global QoS management during processes execution.

Keywords: Service Oriented Architecture, service composition, non-functional constraint, qualitative approach, linguistic approach, multi-criteria decision making, fuzzy Logic, preference modeling, late binding.